UNIVERSITETET I OSLO
Institutt for Informatikk

Hvordan veiledning kan bidra til begrepsbygging innenfor OOP

En eksperimentstudie

Terje Samuelsen

1. April 2007

Hvordan veiledning kan bidra til begrepsbygging innenfor OOP

En eksperimentstudie

En hovedfagsoppgave for cand. scient graden ved
Institutt for Informatikk, Universitetet i Oslo

April 2007

Terje Samuelsen

”Successful efficacy builders do more than convey positive appraisals. In addition to raising people’s beliefs in their capabilities, they structure situations for them in ways that bring success and avoid placing people in situations prematurely where they are likely to fail often. They measure success in terms of self-improvement rather than by triumphs over others.”
(Albert Bandura 1998)

Innholdsfortegnelse

Figurer

Forord

Det å få muligheten til å ta et hovedfag ved universitetet i Oslo, har vært en lærerik opplevelse. Jeg har fått en faglig kunnskap som for meg ville ha vært utilgjengelig uten diverse kurs og arbeidet med hovedfagsoppgaven.

Det mest inspirerende i denne perioden har vært de mennesker som jeg har hatt kontakt med. Både studenter, forelesere, deltagere i prosjekter underveis, samt veiledere.

Jeg vil spesielt ta frem Annita Fjuk (Intermedia), Arne-Kristian Groven (NRL), Jens Kaasbøll, (Informatikk), Richard Borge (informatikk) alle Universitetet i Oslo. Disse deltok i gruppen som planla og gjennomførte vårt eksperiment. Annita Fjuk var den som gjorde det klart at hun støttet min interesse for at muntlig kommunikasjon kunne bidra i læring av OOP. Arne-Kristian Groven bidro med sin ro og hjelp i å kvalitetssikre de to eksperimentene jeg var med på. Han hadde bestandig tid til å bidra. Richard Borge kan jeg best beskrive som ”companions in arms”. Uten hans kunnskap om Karel J hadde eksperimentet blitt noe annet. Når vi satte oss ned og skrev tekst etter prinsippene i parprogrammering, gikk det helt etter malen: to hjerner som utfyller hverandre og stimulerer kreativiteten.

Jeg vil takke gruppen som jobbet med eksperimentet for støtte og veiledning til Richard og meg under planlegging og gjennomføring av eksperimentet. Det å ha samarbeid med personer som delte av sin forskererfaring underveis, ble en inspirasjon for min egen innsats.

Min veileder Jens Kaasbøll har i sin veiledning bidratt som en samtalepartner som jeg kunne avklare ting med. Dette har han gjort uten å ”forelese” og uten å henge seg opp i detaljer. Det jeg leverte etter sommeren fikk en mer dyptgående kritikk samtidig som han beholdt sin rolle mer som veileder enn som coach (se min beskrivelse av veiledning). Takk for det Jens.

Jeg vil også takke Edgar Bostrøm, for å bidra med avklaringer og for å ha lest gjennom oppgaven. Andre som har bidratt med korrekturlesning er: Mette A. Olsen.

Arbeidet startet med mye og interessant arbeid i prosjektet, første bøygen kom da jeg fikk problem med å starte skrivingen. Når jeg kom i gang etter et par måneder fikk jeg bra fremdrift til et visst punkt. Jeg søkte litteratur og leste en mengde. Jeg må i den forbindelse rette en stor takk til bibliotekene ved UIO og Høgskolen i Østfold for all hjelp jeg har fått i mine litteratursøk. De har vist stor tålmodighet når jeg har brukt utover lånetiden.

Noe av det jeg har lest har jeg brukt, men en god del har jeg ikke anvendt direkte med referanse. De har tilgjengjeld bidratt med å gi meg kunnskap og innsikt. Jeg har videre erfart at når mengden med lest litteratur øker, blir det vanskelig å finne de rette sitatene.

Jeg fikk en lang periode som besto av liten fremdrift, og til dels perioder hvor fremdriften stoppet helt opp. Årsaken til dette ligger nok primært i at jeg hadde vanskeligheter med å prioritere tiden som må avsettes til skriving. Når ferien i sommer nærmet seg klarte jeg endelig å få på plass en plan for fullføringen av oppgaven. Vel i gang med ferien ble det skriving, og det var moro å merke at det fungerte. Når grovteksten til et kapittel var klar, ble det dessverre en liten "kryp" periode. Men så ble det lange dager og mentalt følte jeg at jeg begynte å få litt oversikt. Det var en veldig fin følelse.

Til Lone, Håvard, Morten og May.

terje

Sammendrag

Denne oppgaven beskriver og analyserer et eksperiment som ble avholdt som et kurs (2,5 dager) for noen studenter før de begynte ved Institutt for Informatikk ved Universitetet i Oslo. Vi var to hovedfagstudenter som skulle planlegge å gjennomføre kurset som en del av COOL prosjektet etter to hypoteser:

- Se på hvordan grafikk på skjermen kunne virke positivt ved et introduksjonskurs i programmering (Borge, 2004)
- prøve ut om en didaktikk som baseres på veiledning og studentdrevet progresjon, kan bidra til begrepsbygging av objektorienterte begreper.

Min interesse var den andre delen.

Grunnlaget for eksperimentet var at studentene skulle ha en følelse av mestring (Bandura, 1986) etter hvert som kurset skred frem. Metoden som ble anvendt var at vi brukte elementer fra problembasert læring (Duch, hjemmesidene til University of Delaware) sammen med parprogrammering (Sommerville, 2004), samt meget lite skriftlig materiale. Kunnskap skulle bygges på at studentene jobbet med øvelser på en datamaskin.

Som et virkemiddel ble det lagt opp til at dialogen i veiledningen skulle baseres på studentens tidligere erfaringer. For å stimulere studentene til å fokusere på læring av objektorienterte begreper skulle veiledningen rettes mot begrepene og deres anvendelse ved hjelp av tre spørsmål: Hva gjør dere?, hvorfor gjør dere det?, og hvordan hjelper det dere? (Schoenfeld, 1992).

Jeg har valgt å gjøre mine vurderinger ved å fokusere på sentrale begreper innenfor objektorientert programmering, slik som objekt, klasse, sub-klasser, modularisering og gjenbruk.

Viktige funn er at kursopplegget fungerte for bygging av kunnskap om begrepene, å gi studentene mulighet for å prøve ut kunnskapen via å løse problem med programkode.Diskusjonene i gruppene og plenum med aktiv veileder bidro til å bringe frem og avklare oppfattelsen av begreper hos deltagere. Når det gjelder veiledning ved hjelp av de tre spørsmål bidrar disse til at studentene og veileder tidlig får fokus på at det er læring av begrepene som er sentral når du skal lære OOP. Det var overraskende at studentene faktisk tok i bruk spørsmålene når de diskuterte internt i gruppene. Studentdrevet progresjon vil fungere i et kurs som brukes for å introdusere objektorientert programmering. Men det er ikke tilstrekkelig for å sikre at studentene har kunnskap og erfaring nok til å anvende sin kunnskap i mer komplekse oppgaver.

Mangelen på kontrollgruppe og utvalget av studenter var for snever til at resultatet kan generaliseres, men resultatet indikerer at kursopplegget kan være av interesse å prøve i et oppskalert opplegg. I oppgaven presenterer jeg flere problemstillinger, i det vesentlige knyttet til veiledningsfunksjonen, som kan være av interesse for nærmere studier og sammenligninger med andre fagfelt.

1 Innledning

Mange studenter opplever at det å ta sine første programmeringskurs innebærer mye nytt og opplever at de får vanskeligheter på grunn av alt det nye. Et utgangspunkt for meg i denne oppgaven er at vanskelighetene i det vesentlige kommer fra mistilpassing mellom studentenes erfaringsgrunnlag og det undervisningsopplegget som anvendes. I det vesentlige er dagens innledende programmeringskurs basert på en kultur og erfaring i bruk av teknologien, som foreleserne besitter. I det ligger det blant annet at foreleser tar med seg erfaringer fra sin egen studiesituasjon. Men er utvalget av studenter som tar programmeringskurs kanskje annerledes i dag enn for noen år siden? Spørsmål som kan være naturlig: Har antallet studenter økt, er karakternivået hos søkerne lavere, kanskje motivet for å starte på denne type studier har endret seg eller hadde studentene før bedre forutsetninger enn dagens studenter har for å tilegne seg kunnskap som baseres på metoder om overføring av kunnskap fra lærer til student? Dette er et arbeid som ligger utenfor denne oppgaven med bakgrunn i arbeidsmengde.

Jeg kan beskrive min interesse som: Utvikling av kunnskap om hvorledes bygge kunnskap hos studenter som skal lære å programmere datamaskinsystemer som skal driftes i næringslivet. Siden min veileder Jens Kaasbøll var knyttet opp mot COOL prosjektet ble det naturlig at jeg søkte å få et samarbeid mot dette prosjektet.

1.1 COOL

Denne hovedfagsoppgaven er gjort som en del av prosjektet COOL. ”Project COOL (Comprehensive Object-Oriented Learning) was initiated by Kristen Nygaard. In the 1960s Kristen invented, along with his friend and colleague Ole-Johan Dahl, object-oriented ways of thinking through the programming language Simula.” (COOL, Prosjektbeskrivelse). Som det fremgår av teksten springer begrepet COOL ut i fra objektorientert (OO). Dette er et begrep som brukes internasjonalt, og knyttes da opp mot andre begreper som systemutvikling, analyse, design og programmering (OOP). COOL prosjektet forsker blant annet på didaktikken som anvendes i undervisningen av programmeringskurs. “COOL will be limited to research issues related to learning and teaching object-orientation that may have the capacity to improve state of the art education.” (COOL, Prosjektplan). De fleste funn er presentert i boken Comprehensive Object-Oriented Learning: The Learner’s Perspective av redaktørene Annita Fjuk, Amela Karahasanovic og Jens Kaasbøll (COOL, 2006).

I “Work package 2: Learning and teaching programming” (COOL, Prosjektplan) står det at en skal eksperimentere med verktøy og didaktiske metoder spesielt knyttet opp mot to måter å starte læring av objekt og klasser. De to måtene er visualisering av objektenes oppførsel og modellering av objekter i en reell verden. Det fremkommer også i prosjektplanen at der hvor prosedural programmering anvendes ved begynnerkurs, kan studentene legge til seg vaner som må endres når en skal jobbe med objektorientert programmering. Alle disse argumentene er en klar

indikasjon på at det finnes mange grunner for å starte med det objektorienterte paradigmet. Det vil i kapittel 4.7 Programmerings-paradigmer, knyttes mer teori og argumentasjon til denne tesen.

Cool hadde også som mål å se på verktøy som kunne anvendes for å fremme innsikt i objektorientert tankegang i programmeringskurs. Eksempler på verktøy som en har sett på er BlueJ, Robocode og Karel J. Det var meningen å prøve ut flere verktøy for å se på sterke og svake sider relatert til læring om objektorientert programmering. I tillegg var det meningen at prosjektet skulle gi et bidrag til undervisningskompetanse generelt.

Mitt arbeid er gjort i nær tilknytning til følgende deltagere i COOL: Annita Fjuk (Intermedia), Arne-kristian Groven (NRL), Jens Kaasbøll, (Informatikk), Richard Borge (informatikk) alle Universitetet i Oslo. De eksperiment som er lagt til grunn for arbeidet med oppgaven, er en del av COOL og målet har vært at hovedoppgaven skulle prøve å være et bidrag i prosjektet og samtidig være innenfor mitt eget interesseområde.

1.2 Problemstilling

Når det gjelder begynnerkursene for å lære programmering ved universitetene har det vært et meget høyt antall studenter som har droppet kurset underveis eller strøket. Det er gjort flere tiltak for å prøve å rette på denne tilstanden. Blant annet er det prøvd å bruke eldre studenter for å assistere nye studenter ved laboratorium og klasseromsøvinger (Chase og Okie, 2000), la studenter evaluere hverandres kode (Zeller, 2000), programmere i par (Mcdowell et al, 2000), spesiell oppfølging av kvinnelige studenter (Craig, 1998) og av studenter som tar kurset på nytt (Sheard og Hagan, 1999). Alle disse tiltakene har gitt positive resultater, men har ikke ført til de store forbedringene.

Jeg vil her ikke ta for meg alle tiltakene, men heller prøve å fokusere på et tiltak som kan være sentralt for å bedre på den dårlige gjennomføringsprosenten. En av grunnene til at så få gjennomfører denne type kurs kan være at selve undervisningsoppleggene som anvendes ikke er de mest egnete for dagens studenter.

Det er noe av grunntanken i COOL, at studentene allerede i starten på innføringskurs skal lære å forholde seg til objekter. Dette vil lette studentenes arbeid med å bygge kunnskap om de sentrale begreper og der igjennom arbeidet med å lære programmering. Derfor utarbeidet deltakerne i COOL (1.1 COOL) flere eksperimenter. Siden Richard Borge og jeg hadde interesseområder som ville egne seg å studere i et felles eksperiment, ble et av COOL eksperimentene basert på to hypoteser:

- Se på hvordan grafikk på skjermen kunne virke positivt ved et introduksjonskurs i programmering (Borge, 2004)

- prøve ut om en didaktikk som baseres på veiledning og studentdrevet progresjon, kan bidra til begrepsbygging av objektorienterte begreper.

Jeg hadde på et tidlig tidspunkt i hovedfagsstudiet fattet interesse for veiledning. Jeg ønsket å studere nærmere i hvilken grad det er mulig å bygge kunnskap i et introduksjonskurs i programmering bare med støtte av veiledning. Med bakgrunn i de siste års fokus på objektorientert tankegang og den type undervisningsopplegg som Bjarne Herskin (1994) omtaler som "Hans On" undervisning, hadde jeg en spesifikk interesse i eksperimentet. Jeg ville se nærmere på om det var mulig å lære objektorienterte begreper basert på studentens tidligere erfaring, erfaring ervervet med eget programmeringsarbeid og veiledning. Dette er sammenfallende med det som jeg i innledningen (1 Innledning) har beskrevet som min interesse: Å utvikle kunnskap om hvorledes studenter kan lære å programmere datamaskinsystemer.

Med utgangspunkt i dette er problemstillingen i min oppgave: Å prøve ut en didaktikk hvor veiledningen skulle baseres på studentenes eget arbeid med aktiviteter og kursets fremdrift skulle styres av studentdrevet progresjon. For å teste ut i hvilken grad læring fant sted valgte, jeg noen sentrale objektorienterte begreper.

1.3 En skisse av temavinklingen

Før jeg går videre vil jeg gi en kort oversikt over en del begreper som anvendes i denne oppgaven. Det kommer mer utfyllende beskrivelser i kapittel 2 Beskrivelse av begreper som brukes. Begrepet forelesning brukes om en situasjon hvor en lærer forteller og viser stoffet som vedkommende gjennomgår via tavle, lysark eller en projektor. Ordet aktivitet bruker jeg om situasjoner hvor studenten får en problemstilling som er begrenset og skal løse den, enten på basis av stoff som burde være kjent eller med en eller annen form for veiledning. Oppgave bruker jeg om en situasjon hvor studenten får en problemstilling som inneholder en mer kompleks problemstilling og krever et mer omfattende arbeid. Under arbeidet med oppgaver kan studenten som regel få veiledning ved behov, dersom kurset ikke legger begrensninger på hvilken veiledning som kan gis. Eksempel på begrensninger kan være at det skal gis evaluering med karakter på innleveringen, og da ønsker kursledelsen en begrensning på veileders påvirkning på produktet som innleveres.

1.3.1 Programmerings-undervisning

Datamaskiner brukes i mange sammenhenger i dagens samfunn. Og siden de er maskiner må de få instruksjoner om hva de skal gjøre. Disse instruksjonene kalles et program, og må være korrekt i forhold til det som maskinen skal utføre. Programmene som skal brukes, finnes i flere språk og dialekter, men felles for alle er at kravene til korrekthet i syntaks og semantikk er absolutt relatert til det aktuelle språk og dialekt. Dette innbærer at de som skal lære å lage slike program må forholde seg til slike absolutte krav. Stort sett har det å lære seg programmering vært lagt til en postgymnasial utdanning. I de senere år har også denne typen utdanning vært å finne i videregående utdanninger og da gjerne i yrkesorienterte fag knyttet til informasjonsteknologi. På "Workshop on Learning and Teaching Object-orientation – Scandinavian Perspectives Oslo, October 20 2003" uttalte Kølling at i fremtiden ville introduksjonskurs i programmering foregå hovedsakelig på gymnasialt nivå.

Siden min oppgave tar for seg introduksjonskurs i programmering, vil mine problemstillinger også være aktuelle på gymnasialt nivå.

Intensjonen i det første programmeringskurset er ofte å gi studentene en plattform som gjør at de kan utføre programmering i de neste kursene uten å trenge support på det grunnleggende stoffet. Den tradisjonelle metoden for å lære programmering ved universiteter og høgskoler, har vært at kurset bygges opp rundt en lærebok, at det holdes forelesninger og legges opp til det som ofte omtales som øvelser. Disse bygger på stoffet som er gjennomgått i en forelesning og det er vanlig at studentene får hjelp av assistenter når de jobber med disse øvingene. Tiden det jobbes med øvelser på ligger ofte innenfor 10 minutter – 2 uker. Jeg vil gjøre leseren oppmerksom på min bruk av begrepene øvelse og aktivitet som er omtalt i kapittel 1.3.2 Veiledning og 2.8 Aktivitet.

1.3.2 Veiledning

Mitt tema snur problemstillingen til at arbeidet med øvelsene er sentral, for å skape innsikt hos studentene og at veiledningen ikke bare skal støtte opp om arbeidet med øvelsene, men være med på å bygge en forståelse for programmering. For at dette skal skje må veiledningen og arbeid med programmering brukes for å innføre begrepsapparat og gi studenten forståelse av struktur og sammenhenger. Siden studenten skal bygge kunnskap gjennom sitt arbeid bruker jeg begrepet aktivitet. Det at aktiviteter brukes til å innføre begreper, innbærer at arbeidet med aktivitetene legger føringer for det som gjøres i eventuelle forelesninger. I dette ligger det at forelesningene skal ha som bærende element, at de skal støtte opp om og bygge på arbeidet med aktivitetene. Dette gjelder enten forelesningene holdes som egne selvstendige aktiviteter eller er knyttet opp mot aktivitetene som innledning eller oppsummering.

Min interesse er å se nærmere på om det er en didaktikk som kan fungere ved den type veiledning som jeg har beskrevet ovenfor. Dersom denne veiledningen anvender metoder og teknikker som bringer fokus på løsning av den konkrete oppgaven, mer enn på forståelse for sammenheng og begrepsdannelse, vil dette kunne medføre at de aller fleste studentene løser oppgaven, mens flere av studentene ikke har fått mer innsikt som kan overføres til andre oppgaver.

1.3.3 Kommunikasjon

Det er to hovedformer for muntlig kommunikasjon som brukes i undervisning og disse skiller seg vesentlig fra hverandre. Den ene baserer seg på at læreren prøver å formidle hva læreboka beskriver og hva læreren har av kunnskap om emnet. Dette er et syn på læring som baseres på at lærer overfører kunnskap til den som skal lære. Den andre baseres på kommunikasjonen som ligger i et samarbeid mellom studenter og lærer og student. Dette er en form som bygger på studentens tidligere erfaring og kunnskap.

Denne erfaringen er individuell for hver enkelt student. Noen kan være vant med å bruke ordet klasse om en gruppe elever, mens andre bruker det om kvalitet, for eksempel uttrykket ”kvalitet av en helt annen klasse”. Mens lærere som underviser i programmering bruker sin lange erfaring med faget, kollegaer og mengder av litteratur som er lest som grunnlag for sin anvendelse av begrepet. Det er disse forskjellige erfaringer som ligger til grunn for tolkingen som den enkelte anvender i en undervisningssituasjon. Denne forskjellen vil være der så lenge det finnes en undervisningssituasjon. Det er en umulig oppgave å lage et opplegg for undervisning hvor kommunikasjons problemer ikke eksisterer. Derimot er det mulig å forbedre de opplegg som brukes, slik at kunnskapsmålene (Hofset, 1995) blir lettere å oppnå for studentene. Arbeidet i denne oppgaven bruker som utgangspunkt at kommunikasjonen bør situasjonstilpasses til deltagere og tema i større grad enn det som har vært vanlig.

Jeg vil prøve å analysere interaksjon mellom student og lærer og da vesentlig fokusere på den muntlige formen. I dette ligger det at jeg tar utgangspunkt i at språkets anvendelse kan være et bidrag til å lette forståelsen av hvordan de enkelte element i et dataprogram og begreper henger sammen. Denne type kommunikasjon er avhengig av interaksjon mellom to eller flere personer.

1.4 Min oppgave

Når jeg videre i oppgaven bruker begrepet vi, er det de to instruktørene, samt to observatører som var til stede under kurset som omtales. Observatører var Annita Fjuk og Arne-Kristian Groven. Når jeg skriver at vi observerte, så ligger det i det at vi har snakket om det og at det ligger felles forståelse av observasjonen. Her må jeg gjøre oppmerksom på at jeg ikke har beskrevet nærmere hvem som har observert de forskjellige situasjoner. Når jeg bruker begrepet jeg, så er det mine observasjoner eller mine fremføringer.

Med utgangspunkt i kapittel 1.2 Problemstilling, prøver denne oppgaven å dokumentere planlegging, gjennomføringen og resultatene av et eksperiment.

For å gjennomføre veiledningen valgte jeg en veiledningsmetodikk basert på situasjonsbetinget veiledning (2.7 Veiledning) og bruke tre spørsmål (4.6 Veiledningen baseres på tre spørsmål). Siden studentdrevet progresjon er et hovedtema i oppgaven, blir dette nærmere omtalt i kapittel 7.2.3 Aktiviteter skulle baseres på erfaring og 7.2.4 Hver aktivitet skulle tilføre et nytt problem. For lettere å kunne vurdere den studentdrevne progresjonen er kursgjennomføringen (8 Gjennomføringen av eksperimentet) dokumentert kronologisk.

Siden vi to som skulle holde kurset, ikke fant en didaktikk som hadde de forutsetningene vi ønsket, for utprøving av det som er beskrevet i 1.3.2 Veiledning, måtte vi utvikle en didaktikk som kunne fungere for formålet. Til dette ønsket jeg at vi skulle basere oss på elementer som mestring (4.2 Mestre), problembasert læring (4.4 Problembasert læring), parprogrammering (4.5 Parprogrammering) og grafikk på skjermen (1.2 Problemstilling). For å forsterke betydningen av veiledningen ville

vi at studentene i liten grad skulle være avhengig av skriftlige materiale (7.2.2 Skriftlig dokumentasjon skulle være kort).

For å studere utviklingen av begrepsforståelsen valgte jeg å se nærmere på sentrale begreper innenfor objektorientert programmering, slik som objekt, klasse, sub-klasser, modularisering og gjenbruk.

2 Beskrivelse av begreper som brukes

2.1 Student

Student bruker jeg om personer som tar en postgymnasial utdanning. I vårt sommerkurs var deltagerne å betrakte som studenter siden de var tatt opp på universitetet.
I dagens situasjon må vi regne med at studenter som starter på et kurs hvor de skal lære å programmere har varierende bakgrunn. Fra de som bare har anvendt datamaskinen i jobb eller på videregående skole, til de som har drevet med datamaskiner og programmert hjemme på rommet i flere år.

Det å ha faglige utfordringer til personer med en slik spennvidde er en vanskelig oppgave. De som har drevet med programmering har fått vaner og anvender kanskje løsningsmetoder som egner seg for enkle problemstillinger, mens metodene ikke egner seg for mer komplekse problemstillinger. Derfor vil det være sentralt, at studentene tidlig utvikler gode arbeidsvaner og får strukturer å jobbe etter.

En student som skal lære seg programmering, har ofte et ønske om å kunne anvende sin kunnskap i emnet til å løse en oppgave som vedkommende ønsker løst. Denne oppgaven kan være å styre en prosess i industrien, lage et program som kan gi en pen webside, lage et program for et idrettslag, få utstyret på datamaskinen til å fungere bedre, lage et spill som er slik en ønsker og så videre. Disse oppgavene oppleves nok av studenten som praktiske oppgaver som har en nytteverdi og de kan derfor kanskje motivere for arbeidet som må gjøres. Derimot kan oppgaver som studenten opplever som mindre nyttig, gi en lavere motivasjon for arbeidet som må gjøres i forbindelse med læringen. Med uttrykket mindre nyttig mener jeg her at studenten ikke ser noen praktisk anvendelse som kan være nyttig for vedkommendes interesse. Temaet motivasjon kommer jeg tilbake til under teorikapitlet.

2.2 Tradisjonell undervisning

Tradisjonelt har universiteter og høgskoler mye basert seg på å formidle kunnskap ved å anvende en lærebok. På universitetene er ofte lærebøker som brukes i de innledende programmeringskurs skrevet av foreleser eller av andre på samme fakultet. Min erfaring som er i overensstemmelse med Kölling (2005) er at mange lærere følger bokens tekst rimelig tett når de gjennomfører sin undervisning.

Faget er i såpass rask utvikling at en bok ikke har så lang tid den kan anvendes. Derfor må opplagsvolumet være såpass at det er verdt innsatsen å skrive og utgi boken. Siden universiteter ofte kan være med å bidra til antall potensielle kjøpere er det ofte denne type bøker som også brukes på høgskoler.

Det stoffet som en skal gjennom i et innledende kurs er ofte så omfattende og krever såpass mye øvelser at det blir en avveining om hvilke temaer som skal behandles med forelesning, øvelser eller oppgaver. Det er en svakhet at tiden foreleser har til disposisjon sjelden gir anledning til en grundig gjennomgang av stoffet samtidig som det skal gis eksempler. Foreleser havner derfor ofte i et dilemma, vedkommende kan gjennomgå stoffet i konsentrert form ved å fokusere på det som læreren oppfatter som viktigst. Dersom vedkommende fokuserer på sammenhengen med det øvrige stoffet, kan ofte det aktuelle tema bli for overflatisk behandlet. Denne avveiningen er vankelig å få til og ofte brukes det egne timer til øvelser som et supplement. Disse øvelsene bygger på stoff som er gjennomgått teoretisk på forelesning og har som regel innlagt en del forutsetninger når det gjelder forberedelser og forståelse. Med forståelse menes her lærerens forståelse av hva som er viktig/sentralt i stoffet og da med bakgrunn i vedkommendes erfaring.

Forståelsen bygger nødvendigvis ikke på studentens kunnskap og erfaring og det kan derfor bli undervisning på feil grunnlag. Når studenten senere skal anvende sin kunnskap vil misforholdet vanskeliggjøre den ønskede læringen.

Før ble det regnet som helt naturlig at en brukte nedenfra-opp tilnærming til datateknologien. Dette ble ofte gjort ved at en startet med å lære hvorledes en skulle skrive programmer med riktig syntaks på en datamaskin, for så å sette sammen funksjoner som kunne utføre en oppgave vi ønsket utført. Så gikk en videre til å lære hvorledes programmene kunne samarbeide. Denne tilnærmingen gjenspeilte seg også i undervisningsoppleggene, som startet med å fokusere på syntaksen i programmene for så å jobbe videre med hvorledes en skulle lage programmer med komplisert semantikk.

Dette at en før ofte brukte en nedenfra-opp strategi i oppbyggingen av kunnskap stemmer jo bra med bakgrunnen til de som skulle lære. De hadde forholdsvis liten innsikt sammenlignet med dagens studenter til hva datamaskiner kunne anvendes til. Det er derfor kanskje naturlig at en startet med et grunnleggende element som programsyntaks, for så å bygge opp kunnskapen til å se elementer i større sammenhenger. Men dagens unge har en helt annen erfaring i anvendelse av media og datamaskinbasert teknologi. Det har jo skjedd en voldsom endring ved at veldig mange har Internett forbindelse hjemme, skolene har maskiner med Internett forbindelse, bruk av mobilteknologi, spill og medias bruk av teknologien.

Medias anvendelse av teknologi synliggjør mange av mulighetene som ligger i anvendelse, både ved å anvende den i presentasjoner og i program som utdyper litt hvordan teknologien anvendes. Dette gir jo de personene som skal lære programmering på et universitet eller høgskole en allsidig bakgrunn og mulighet for å ha opplevd bruk av teknologien, før de kommer til det punkt hvor de skal lære å programmere i undervisningssammenheng.

Denne tilnærmingen til teknologien kan defineres som ovenfra-ned prinsippet. I det legger jeg: At vi gjennom anvendelse, for eksempel via en skjerm ser hva som skjer og etter hvert danner vi oss kunnskap om samspillet mellom funksjoner. Dette kan vi så bruke som grunnlag for å prøve å forstå hvorledes datateknologien er oppbygd og videre hvorledes datamaskiner fungerer.

Pedagogiske opplegg innenfor informatikkundervisning er ofte et resultat av tradisjoner. Dette kan komme av at undervisningen er nært knyttet til en lærebok (dette utdypes senere). Den største vanskeligheten med å prøve ut et annet pedagogisk opplegg enn en bok legger opp til, er trolig tradisjoner som finnes på den enkelte institusjon. Enhver undervisningsinstitusjon utvikler etter hvert sine egne tradisjoner.

Selv undervisningspersonal som kommer utenfra, har tatt sin utdannelse ved en akademisk institusjon, og det er derfor naturlig at det finnes felles trekk i vurderinger som brukes for å vurdere utdannelse i programmering. Når et nytt kurs utvikles, snakker en gjerne med sine kollegaer på institusjonen. Dette medfører at det ligger stort sett samme erfaringsgrunnlag til grunn for utarbeidelsen av et nytt kurs. Dette behøver ikke være negativt. Siden studentene ofte skal ta flere kurs som bygger på kunnskapen fra det første kurset, kan det bli vanskeligheter dersom de neste kursene bygges opp etter helt andre pedagogiske prinsipper. At opplegg for kurs følger de samme tradisjoner kan være en fordel for studenter i det de kan gjenkjenne malen som kursene er bygget opp over.

Mennesker snakker ofte om tradisjoner og vi vurderer tradisjoner ut i fra hva vi selv har erfart. Dette at læreren anvender sin egen erfaring når undervisningsopplegg skal vurderes er i seg selv ikke negativt. Det som kan være lite heldig, er at læreren bruker som forutsetning det vedkommende selv opplevde da han/hun tok sitt første programmeringskurs. Dersom vedkommende opplevde at dette var bra, og bruker denne erfaringen på dagens studenter, blir da spørsmålet om dagens studenter har den samme bakgrunnen som læreren hadde da vedkommende startet sin utdanning. (Forskjell i bakgrunn omtales i eget avsnitt.)

Siden dagens studenter har mer innsikt i hva teknologien brukes til, men fortsatt ikke har så mye innsikt i programmering av datamaskiner, kan det kanskje være naturlig å anvende den kunnskap og erfaring som de besitter som utgangspunkt for å lære programmering? Dette medfører en helt annen strategi i undervisningsopplegget enn det tradisjonelt har vært brukt ved høgskoler og universitet.

2.3 Språk

Språk kan i hovedsak brukes for å uttrykke seg ved hjelp av skriftlig tekst eller muntlig. I undervisningssammenheng blir det i de aller fleste sammenhenger brukt det språket som institusjonen naturlig sogner til. Det medfører at de fleste studentene har kjennskap til hvordan språket anvendelse i muntlig og skriftlig form. Men det er viktig å være oppmerksom på at kunnskapen om språket, varierer både hos de som har språket som morsmål og ikke minst hos de som kommer fra en annen etnisk bakgrunn. Når språket skal brukes som et hjelpemiddel for å bygge opp kunnskap i et nytt begrepsapparat for studentene, er det viktig at en bruker kjente erfaringer og begreper som grunnlag.

I daglig tale vil ofte et litt unøyaktig språk ha lite å si for forståelsen til mottakeren av det som vi ønsker å uttrykke. Dette er vi vant med helt fra barna er små, da må vi tolke det som de prøver å uttrykke. Da blir vi vant med at små unøyaktigheter i

språket stort sett er greit. Etter hvert som vi blir eldre og kommer oppover i skoleklassene skjerpes kravene til nøyaktighet, men forsatt er det rom for å uttrykke seg med ganske stor grad av frihet i språket. Dette kommer også til uttrykk i at vi i enkelte sammenheng vurderer språk som muntlig språk og skriftlig språk. En elev på ungdoms eller videregående skole skal ikke levere inn skriftlige besvarelser med en språkdrakt som gjenspeiler det muntlige språket som anvendes i klasserommet og utenfor. Denne forskjellen tyder på at vi har forskjellige kriteria når vi vurderer språket som anvendes. Det at vi skiller på kriteria er ikke implisitt at det skriftlige språket alltid har en stor grad av nøyaktighet og korrekthet i forhold til ønsket semantikk. Det uttrykker snarere en grov inndeling av de krav som brukes i vurderingen.

En årsak til at unøyaktighet i daglig tale kan gå greit, er at vi anvender all vår tidligere kunnskap og erfaring når vi som mottaker danner oss en oppfattelse av det som vi tror er semantisk korrekt. Dette leder til at muntlig kommunikasjon mellom personer som har samme kulturelle bakgrunn, gir en større mulighet for å korrigere forståelsen av det som blir sagt. Dette leder igjen til at kommunikasjon med personer med annen kulturell bakgrunn krever en mer omstendig beskrivelse av temaer. Det er verdt å merke seg at selv da, vil en formulering være basert på forfatterens kulturbakgrunn.

Et eksempel på at et begrep i en kultur kan oppfattes på en måte mens det i en annen kultur betyr noe ganske annet, er ordet prototype. I hovedfagskurset ”Systemarbeid: Teori og systemer” opplevde vi at begrepet kunne ha forskjellig betydning. Vi fra UIO brukte ordet prototype, om et første utkast til system eller del av et system som vi skulle videreutvikle før det ble levert til kunden. Mens de fra Washington State University (WSU) brukte ordet om den første leveransen til kunden. Det er verdt å merke seg at alle personene var studenter, og kom fra industrialiserte land. Dersom kulturforskjellen er mer sprikende må en påregne større problemer knyttet til forskjellig oppfattelse av språket som anvendes.

Vi må påregne at nye studenter som kommer til et universitet har liten erfaring fra språkkulturen på det aktuelle universitet, samt at det er kulturforskjeller mellom studentene. Dette på bakgrunn av at de kommer fra forskjellige skoler, deler av landet, verdensdeler og etnisk bakgrunn og dermed har forskjellig erfaringsgrunnlag.

Forskjell i bruken av språket finnes også knyttet til aldersforskjeller. Men språklige vanskeligheter med bakgrunn i aldersforskjell mellom lærerer og studenter på universiteter og høgskoler vil innenfor mitt domene i større grad være basert på forskjell i kunnskap og erfaring, og i liten grad basert på aldersforskjellen i seg selv.

2.4 Erfaringsbakgrunn

Derimot kan aldersforskjell mellom lærer og student gi grunnlag for mangel på innsikt i studentenes erfaringsbakgrunn og interesser. Dersom det ikke tas hensyn til denne mangelen på innsikt i undervisningsopplegget er det lett for at opplegget baserer seg på eksempler som studentene ikke har tilstrekkelig bakgrunn for å

anvende. Dette medfører at det forberedne arbeidet må innebære mer undersøkelse av hvilken bakgrunn og erfaringer studentene som kurset retter seg mot har. Siden utdanning som tas før universitet og høgskole for tiden, endrer seg mye i Norge når det gjelder bruk av datamaskiner i undervisningen, vil dette bidra til å forsterke endringer i erfaring og bakgrunn til studentene som starter på programmeringskurs.

De pedagogiske opplegg har også endret seg kraftig i disse typer av skoleslag. Ved at det allerede i starten på skolegangen vektlegges at elevene skal prøve å uttrykke helhet og forståelse. Et eksempel på dette er måten å lære å skrive norsk på som har endret seg kraftig. Før lærte en å skrive ordene riktig for så å prøve å uttrykke noe med å skrive setninger. Nå lærer de å skrive en tekst med mening, men det godtas at syntaksen er basert på fonetiske lyder og er feil i forhold til rettskrivingsregler. Først senere legges det opp til å tilnærme seg korrekt syntaks og semantikk. Denne tilnærmingsmåten er å sammenligne med det som jeg tidligere har beskrevet som ovenfra-ned. Denne tilnærmingen brukes i liten grad på det pedagogiske opplegget som brukes i tradisjonelle programmeringskurs.

Elever som blir studenter og skal starte sin postgymnasialutdanning har stort sett rik erfaring fra en undervisningssituasjon hvor det er stor tilpassning til erfaringer og bakgrunn. Det kan være at høgskoler og universitet ikke har tatt tilstrekkelig hensyn til dette i utarbeidelsen av sine kurs. Spesielt vil tilpassing til erfaring og bakgrunn være vanskelig i skriftlig materiale som er ferdig laget. Derimot, når mennesker kommuniserer med bakgrunn i skriftlig materiale, vil det gi mulighet for å kunne korrigere kommunikasjonen underveis, ut i fra de erfaringer som gjøres.

Programmene som styrer hva datamaskiner skal utføre, har ikke samme mulighet for korreksjon som mennesker har når de skal anvende en tekst. Maskiner er absolutte, enten virker det eller så virker det ikke. Dette innbærer at syntaks og semantikk må være korrekt uttrykt i forhold til det programmereren ønsker skal skje under utførelsen av programmet.

Undervisningsopplegg brukt til å lære programmering på universiteter og høgskoler har ofte vært basert på det som tidligere er beskrevet som nedenfra-opp struktur. Dette stemmer bra overens med den erfaringsbakgrunn som de som holder kursene har. De lærte seg språket først og så lærte de å uttrykke noe om hvorledes de skulle få datamaskinen til å gjøre noe som hadde en praktisk nytte. Denne metoden å legge opp utvikling av kunnskap, stemmer lite med hvordan vi lærer oss et talespråk slik det er beskrevet tidligere.

Disse forskjellene i bakgrunn mellom undervisningspersonalet og nye studenter burde indikere at det blir tatt hensyn til forskjellen, når det utarbeides nye undervisningsopplegg.

2.5 Bøker

Bøker som brukes i forbindelse med undervisning i programmering ved universiteter og høgskoler prøver å ha et mest mulig nøyaktig språk og det vektlegges at definisjoner er nøyaktig. Dette er en naturlig følge av at det i de akademiske miljø

har utviklet seg en kutyme for hvilke krav som en forventer skal oppfylles med hensyn på korrekthet i språk og muligheter for å kunne dokumentere påstander i en tekst.

Bøkene har ofte en omstendig form for å tilfredsstille de akademiske krav til nøyaktighet og korrekthet. Dette sammen med kravene til korrekthet i syntaks og semantikk, bidrar til at fokus lett blir på det som skal til for at programmer kan kjøres på en datamaskin. Denne fokus gjelder både lærer og studenter som skal tilegne seg kunnskaper om å skrive programmer. I tillegg til at bøkene skal brukes i en undervisningssituasjon, skal de kunne fungere som selvstendig dokumentasjon og dette vil også legge føringer til form og omstendighet i behandlingen av de enkelte temaer.

For at en bok skal fungere som selvstendig dokumentasjon for å lære å programmere, vil det være behov for at den har et høyt presisjonsnivå i forhold til syntaks og semantikk i programmene. Dette krever at denne type stoff tar en del plass, og det kan lett virke unaturlig dersom andre deler er mer unøyaktig og har andre krav til nøyaktighet i språk og beskrivelser. Disse kravene til bøker, gjør at den situasjonstilpassning som muntlig språk gir muligheter for i liten grad er til stede.

Bøker som brukes for å lære programmering, starter ofte med et enkelt program og en beskrivelse av hvorledes semantikken i programsetningene skal være bygget opp (5 Vurdering av bøker).

Den type oppgaver som brukes i starten av bøkene er ofte bygget opp ved at det skal skrives noe på tastaturet og så skal et program skrive ut denne teksten på skjermen. Neste steget er ofte en utvidelse som består av at det som skrives inn behandles av en enkel algoritme før det vises noe på skjermen. Eksempel på dette er å legge sammen tall. Denne typen øvelser kan lett oppfattes som en øvelse for å lære noe teori samtidig som studentene har vanskelig for å assosiere dette med å løse en praktisk oppgave.

Først lenger ut i bøkene kommer det stoff hvor det skjer noe som en student lettere kan oppfatte som en praktisk anvendelse. Begreper som regnes som sentrale for å drive programmering kommer ofte lenger ut i bøkene, et eksempel på dette er klasser, objekter, modulisering og gjenbruk.

Selv om det istedenfor bøker brukes kompendier får disse lett samme form som bøker. Årsaken er kanskje at kompendier ofte blir utviklet i samme tradisjon som bøkene, og ofte er første utviklingsstadiet for en bok. Forfatterne kan lett gå i en felle, siden de først bruker kompendiet og forbedrer det frem til en bok. Fellen er at det kan bli en oppfattelse av at resultatet av denne arbeidsprosessen nødvendigvis blir et godt pedagogisk opplegg for et programmeringskurs.

Ofte burde det ha vært gjennomført en mer kritisk evaluering av det undervisningsopplegget som boken legger opp til, versus opplegg som er basert på helt andre pedagogiske prinsipper.

2.6 Forelesning

Forelesning brukes i denne oppgaven om en situasjon hvor studentene får en gjennomgang av teorien i stoffet. Gjennomgangen kan godt inneholde eksempler og emner som knytter stoffet sammen med andre fag.

Som beskrevet i 2.2 Tradisjonell undervisning, blir innholdet i en bok ofte førende for de undervisningsopplegg som anvender boken. Siden forelesningene er såpass sentral i et tradisjonelt undervisningsopplegg er det naturlig at disse også blir sterkt påvirket.

Dette kan i det vesentlige komme av to grunner. En er at foreleser har skrevet boken selv, den andre er at vedkommende ønsker å gjøre det lettere for studentene å lese stoffet. Det er også enklere å lage forelesninger som i stor grad følger en bok sin progresjon i stoffet (Kölling, 2005). Dersom det i tillegg brukes de samme øvelsene og begrepene, vil forelesningene være mer eller mindre en kopi av boken. Selv om en bruker en annen progresjon enn boken, det vil si ved å hoppe frem og tilbake i boken, vil en ofte bruke de samme eksemplene og vektlegge de samme momenter som forfatteren har lagt opp til. Konsekvensen av dette kan være at kursene som bruker boken, anvender den samme pedagogikken. Dette behøver ikke nødvendigvis være noen suksess.

En vanskelighet med den tradisjonelle formen for forelesning slik den er beskrevet ovenfor, er at den ofte blir bundet i formen og gir liten mulighet for situasjonstilpassning. Dersom studentene har en annen bakgrunn og erfaring (2.4 Erfaringsbakgrunn) enn det forlesers opplegg forutsetter, så kan en vanskelig påregne den progresjon hos studenten som opplegget legger opp til. Nå kan det brukes som argument at med en stor gruppe på forelesning er det umulig å situasjonstilpasse forelesningene. Dette kan stemme til en viss grad, men da burde det være mulig å gjøre litt grundigere undersøkelser på studentenes bakgrunn for å få et mer tilpasset opplegg. Et annet alternativ er å finne frem til opplegg som gir større mulighet for mer situasjonsbetinget tilpassning. Dersom forelesningene var mer selvstendige produkter som i mindre grad ble styrt av en bok, kan det bli enklere å tilpasse forelesninger og øvelser slik at læring ble mer et resultat av samarbeid (Herskin, 1994).

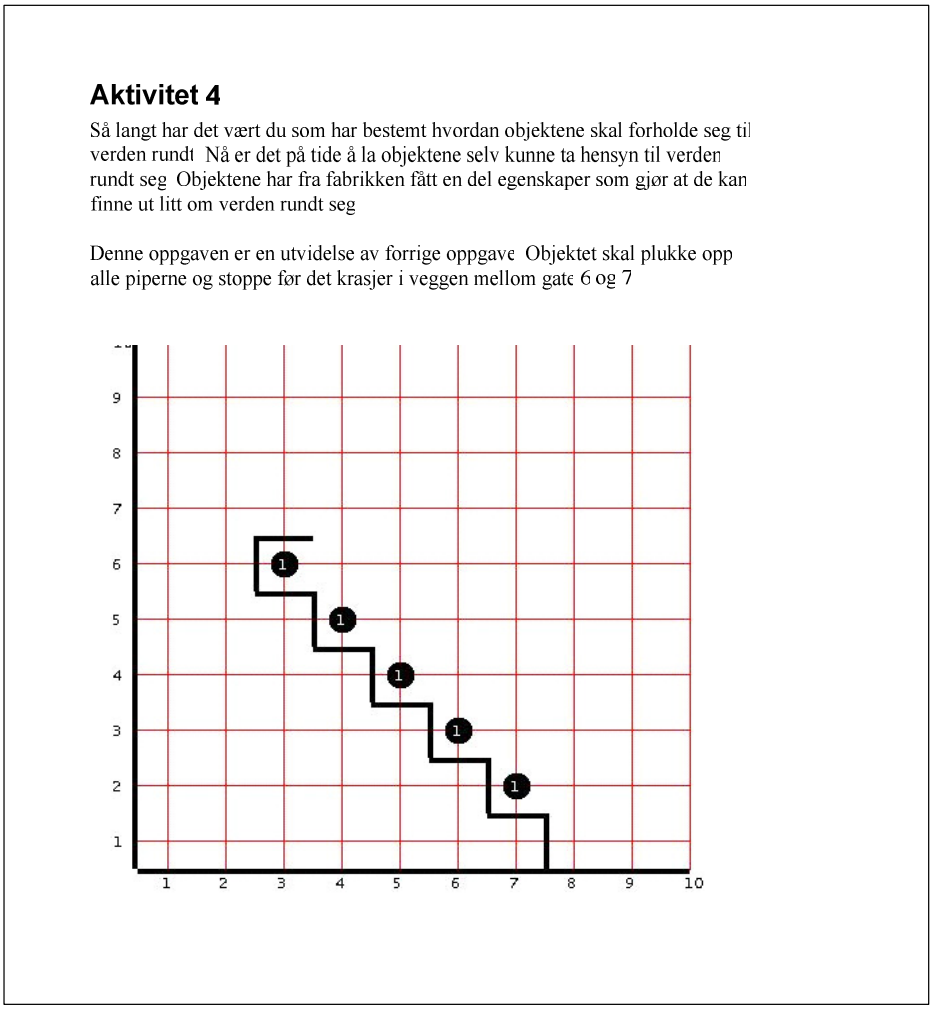
2.7 Veiledning

Begrepet veiledning brukes om det arbeid som gjøres av en instruktør eller undervisningsassistent og rettes mot en person eller gruppe som er i en læresituasjon. Læresituasjonen er her arbeid med å løse et problem, omtalt i denne oppgaven som aktivitet (2.8 Aktivitet) eller oppgave (2.9 oppgave). I dette eksperimentet er veiledningen basert på den verbale kommunikasjonen som foregår mellom veileder og student når studentene jobber med å løse et problem. Jeg inkluderer også bruk av illustrasjoner som kan være både tekst og figurer.

Dersom veiledningen i stor grad tar utgangspunkt i at den skal tilpasses situasjonen, omtales det i denne rapporten som situasjonsbetinget veiledning. I dette ligger det at veiledningen i størst mulig grad skal tilpasses studentens kunnskap og erfaring.

2.8 Aktivitet

Ordet aktivitet kan i vårt domene beskrives som et arbeid som studenten må utføre ut i fra en tekst som beskriver en problemstilling innenfor det aktuelle domenet. Arbeidet skal være begrenset både når det gjelder arbeidsmengde og problemstilling. Eksempel på aktivitet kan typisk være: Mindre enn to timers arbeid og bare omhandle et nytt læringsmoment/problemstilling. Aktiviteter brukes til å innføre nye begreper og stoff og det må derfor være en innledning som tar opp disse nye ting. Et typisk eksempel på hva vi bruker begrepet aktivitet om er Aktivitet 4:



Figur 1: Eksempel på aktivitet

2.9 Oppgave

Oppgaver kan i vårt domene defineres som aktiviteter som krever en større arbeidsinnsats enn de som jeg omtaler som aktivitet. De inneholder ofte en mer kompleks struktur på problemstillinger og kan inneholde flere læringsmomenter. En typisk oppgave er en arbeidsoppgave som skal leveres inn for godkjenning, eventuelt karaktersetting.

3 Metode

Dette kapitlet er bygget opp ved at første del er en er en presentasjon av forskningsmetoder som ofte brukes innen forskning på informasjonsteknologi og undervisning i informatikk. Det også en avklaring av min anvendelse av en del begreper som brukes i forbindelse med beskrivelser av metoder i forskningen. Delen består av kapittel 3.1 Metode -presentasjon, 3.2 Kvantitativ tilnærming, 3.3 Kvalitativ tilnærming og 3.4 Kvantitativ versus kvalitativ tilnærming.

Den andre delen beskriver mitt valg av metoder og består av kapittel 3.5 Mitt valg av metoder, 3.6 Valg av tilnærming og 3.7 Innsamling av data.

3.1 Metode-presentasjon

Vanligvis vil det på et tidlig stadium under idearbeidet med et forskningsprosjekt avklares om undersøkelsen skal ha en induktiv eller deduktiv arbeidsform. Halvorsen (2003) definerer **"Induktiv metode,** vitenskapelig metode som bygger på slutninger fra enkeltobservasjoner til formulering av en teori eller hypotese om generelle sammenhenger." Setter jeg det på spissen vil det si at vi kan starte med observasjoner uten å ha noe mål for hvilke teorier vi kan finne. Den andre arbeidsformen tar et motsatt utgangspunkt ved "Research designed to demonstrate a hypothesis is deductive..." (Hoyle et al. (2002).

En årsak til at det finnes flere metoder innenfor forskning er at enkelte metoder er mer egnet en andre for å finne frem til hva vi ønsker å jobbe med. Siden poenget med valg av metode må være bestemt av hva vi ønsker å finne frem til, ble det også førende i vårt valg av metoder.

I boken "Qualitative Research in Information Systems"(Myers og Avison, 2002, red), skriver forfatterne at forskningsmetoder kan bli klassifisert på forskjellige måter. Men at en vanlig måte å skille på, er mellom kvantitativ og kvalitativ tilnærming. Siden forskning fra gammelt av har bygget opp en tradisjon for å anvende kvantitativ tilnærminger, starter jeg med å beskrive disse. Med bakgrunn i at vi nedprioriterte denne tilnærmingen vil beskrivelsen være kort.

3.2 Kvantitativ tilnærming

Kvantitative forskningsmetoder var opprinnelig utviklet for å studere naturvitenskaplige fenomen. Eksempel på kvantitative metoder som nå er akseptert innenfor sosialvitenskapen er laboratorieeksperiment, formelle metoder og numeriske metoder (Myers og Avison, red. 2002). Det at kvantitative metoder ble utviklet for å studere naturvitenskapelige fenomen er kanskje naturlig siden det i den forbindelse ofte er snakk om kvantitative beskrivelser av fenomen. Dette finner jeg

godt i overensstemmelse med definisjonene til Grønmo (2004) og Halvorsen (2003) om at kvantitative data, er data i form av tall eller andre mengdetermer. Men kvantitativ tilnærming dreier seg i liten grad om datakildene, men heller om hvilke metoder som anvendes for å behandle disse dataene.

Slik jeg tolker dette, er at en kvantitativ tilnærming medfører at det kan presenteres et resultat eller deler av et resultat ved hjelp av tall eller mengdetermer slik Grønmo og Halvorsen omtaler ovenfor.

For at en kvantitativ tilnærming skal kunne gi resultater som kan ha en sterk validitet som forskningsmessig resultat, bør datagrunnlaget være basert på et så brett grunnlag at det kan sies å være representativ for tilsvarende gjennomføringer eller at det kan generaliseres. Dette kommer jeg tilbake til senere.

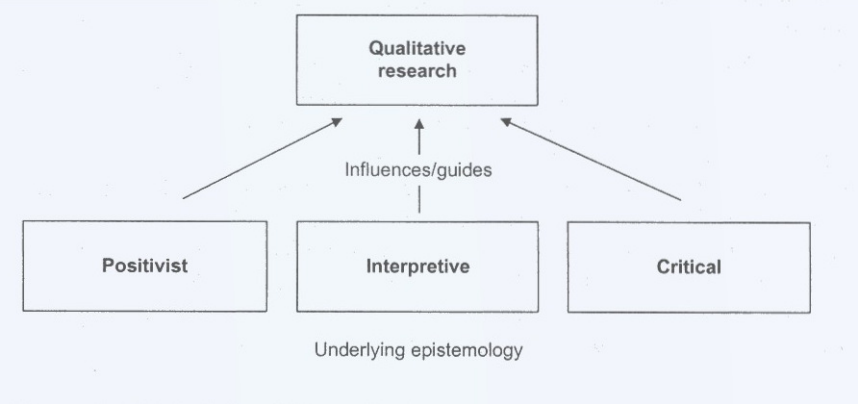
3.3 Kvalitativ tilnærming

Kvalitative forskningsmetoder var utviklet i sosialvitenskapen for å gjøre det mulig for forskere å studere sosiale og kulturelle fenomen. De er designet for å hjelpe oss å forstå folk i den sosiale og kulturelle konteksten de lever i. Kvalitative datakilder inkluderer observasjon og deltagende observasjon (feltarbeid) intervjuer og spørreundersøkelser, dokumenter, tekster og forskerens inntrykk og reaksjoner (Myers og Avison, red. 2002).

En annen beskrivelse er fra Hoyle *et al.* (2002): "…**qualitative research:** Instead of researchers imposing their own hypotheses, themes, and categories on the participants` responses, the participants relate stories about their lives that enable the researcher to generate hypotheses and themes." Denne beskrivelsen vil jeg hevde illustrerer godt at forskerne gir mulighet til deltakerne å presentere sin versjon/reaksjon og at forskeren tolker dette ut i fra sin egen kunnskap og bakgrunn. For å illustrere forskjellen på kvantitativ og kvalitativ tilnærming kan jeg ta utgangspunkt i at det i en undersøkelse blir brukt spørreundersøkelse. Da vil det statistiske materialet representere en kvantitativ tilnærming, mens dersom forskerne gjør vurderinger, som ikke direkte kan leses uti fra det statistiske materialet representerer dette en kvalitativ tilnærming.

3.3.1 Perspektiv

I følge Myers og Avison (red. 2002) kan kvalitativ forskning beskrives med hjelp av filosofiske perspektiv.



Figur 2: Kvalitativ forskning i tre filosofiske perspektiv

Fra: Qualitative Research in Information Systems, 2002

Jeg tar en kort oppsummering på disse tre perspektiv.

Positivistisk også omtalt som objektiv, brukes om en vitenskapsforståelse hvor vitenskapelig kunnskap bygges opp gjennom systematisk undersøkelse av positivt gitte eller faktisk eksisterende fenomener. Det er en forutsetning at disse fenomenene finnes som observerbare objektive forhold som kan måles, og at fenomenene ikke blir påvirket av at de observeres og undersøkes. Jeg vil påpeke at det finnes flere oppfatninger av positivisme. Blant annet at samfunnsvitenskap ikke skiller seg fra andre vitenskaper og derfor bør anvende seg av de samme opplegg og metoder som naturvitenskapen. Mens kritikere av positivismen fremfører at samfunnsvitenskapen er så vesensforskjellig fra naturvitenskapen at samfunnsvitenskapen må bygge på metoder som er prinsipielt forskjellig fra naturvitenskapen (Grønmo, Sigmund, 2004). Positivistiske studier brukes generelt ved at det lages en hypotese og så brukes det kvantitative metoder for å teste hypotesen. (Myers og Avison, red. 2002, Fincher og Petre, red. 2004)

Interpretive forskning (tolkning) også omtalt som subjektiv, prøver generelt å forstå fenomen gjennom den mening som folk uttrykker om dem. ”Interpretive studies assume that people create and associate their own subjective and inter-subjective meanings as they interact with the world around them.” (Orlikowski og Baroudi, 1991).

Kritisk forskning forutsetter at sosial virkelighet er historisk dannet og at den er produsert og reprodusert av folk. Kritisk forskning fokuserer på opposisjon, konflikter, indre konflikter i samtiden (Myers og Avison, red. 2002). Dette opplever jeg som lettere å forstå når jeg holder det opp mot: ”…test som går ut på at to teorier stilles opp mot hverandre som innbyrdes motstridene forklaringer på bestemte fenomener i samfunnet, og at minst en av de to teoriene må forkastes som følge av den empiriske analysen av disse fenomenene” (Halvorsen, 2003).

De siste 15 årene har det vært tilbakegang for den positivistiske retningen i vitenskapsfilosofien. I dag foregår det en forskyvning mot filosofiske tenkemåter som ligger nærmere de humanistiske fag (Kvale, 2001). Altså mot et mer interpretive perspektiv.

Nå er det slik at en i samme forskningsprosjekt kan velge mellom det kvantitative og kvalitative prinsipp, men som omtalt ovenfor, det ene utelukker ikke det at en kan kombinere dette valget med også å bruke det andre prinsippet.

3.3.2 Metoder

Kvalitativ forskning benytter seg av flere forskningsmetoder. Slik sier Harvey og Myers (2002) det: ” Research methods need to be both relevant and rigorous in order to be accepted as legitimate within a particular field of knowledge”.

Det å velge en passende metode handler om å finne metoder som er passende i forhold til den aktuelle problemstillingen. Dette er komplisert, blant annet siden det finnes et antall metoder som hver har flere varianter av definisjoner. Jeg skal prøve å gi en kort oversikt over de metoder som jeg har funnet anvendes mest i forbindelse med forskning på informasjonssystemer og undervisning i informatikk. Det kan stilles spørsmål ved at jeg velger et slikt utgangspunkt når det er læring av objektorienterte begreper som er mitt tema. En av årsakene til dette er at jeg har en praktisk tilnærming ved at studentene skal erfare gjennom å jobbe på datamaskiner og derigjennom bygge kunnskap om objektorienterte begreper. Jeg prøver derfor å bruke beskrivelser av metoder som passer innenfor mitt domene.

Aksjonsforskning

Denne type forskning henspeiler på at forskeren er deltagende i arbeidet. Ideen i aksjonsforskning er at det først gjøres en undersøkelse av situasjonen, før det lages en hypotese om endringen en ønsker å undersøke. Så gjennomføres det et arbeid med å få til endringen og effekten av dette studeres (Baskerville og Wood-Harper, 2002). I det ligger det at forskeren ikke er en uavhengig observatør, men at vedkommende deltar aktivt og at endringsprosessen er tema for forskningen. Det er et dilemma at forskeren kan bli for deltagende og dermed redusere objektiviteten som forskningen er avhengig av.

Metoden omtales som ideelt egnet for å studere teknologi i en menneskelig kontekst. (Baskerville og Wood-Harper, 2002). Men som forfatterne påpeker, det er motstridende syn på dette. Eksempel på det er at det sjelden er artikler fra Nord-Amerika som bruker aksjonsforskning i sitt arbeid med informasjonssystemer. Videre finnes det flere artikler som kritiserer anvendelsen av metoden i forbindelse med denne type systemer. På tross av denne kritikken er metoden blitt akseptert som gyldig forskningsmetode knyttet til arbeid med utdanning.

Case-studier

Ideen i casestudier er å undersøke fenomen i en sammenheng som forekommer i det virkelige liv. Metoden er spesielt egnet når vi er avhengig av en naturlig setting eller samhandling mellom hendelser. Dette gjelder særlig dersom grensen mellom

fenomenet som skal studeres og omverden er uklar (Myers og Avison, red. 2002, Fincher og Petre red. 2004). Selv om det er likhetstrekk mellom casestudier og aksjonsforskning er det verdt å merke seg forskjellen mellom dem.

Case-studier er lite egnet dersom det er behov for å kontrollere hendelser eller å påvirke disse under prosjektet. Metoden er godt egnet til å forske på spørsmål knyttet til anvendelse av informasjonssystemer. Dette siden bruken av systemene påvirkes av settingen rundt, det vil si de mennesker og det samfunnet disse opererer i, samt at menneskene og samfunnet påvirkes av hva systemet utfører (Myers og Avison, red. 2002).

Etnografi-studier

Denne type studier stammer fra antropologien og er basert på at forskeren skaffer seg innsikt ved å tilbringe mye tid knyttet til problemområdet (Myers og Avison, red. 2002). Som i antropologien er det særlig de sosiale perspektiv som er i fokus. Hoyle et al. (2002) beskriver denne type studier som relativt ustrukturert. En illustrasjon fra samme bok, på hvordan studiene kan oppfattes er: ”Ethologists attempt to enter a scene to discover what is there.” Beskrivelse av et fenomen gjøres ved å sammenligne med fenomener i en verden som for leseren bør være kjent. Denne beskrivelsen inneholder i utgangspunktet ingen kvantitativ analyse. Metoden er spesielt egnet for å beskrive sammenhengen mellom elementer, som for eksempel et datasystem og den kontekst disse brukes i.

Den nærheten som oppstår når en forsker bruker så mye tid i problemområdet gjør at det kan stilles spørsmål med objektiviteten til forskningsarbeidet.

Grounded teori

Harvey og Myers (2002) beskriver teorien slik: ”Grounded theory is a research method that seeks to develop theory that is grounded in data systematically gathered and analysed.” Videre sier de at metoden anvendes i økende grad innenfor litteratur om forskning på informasjonssystemer. Dette fordi metoden er ekstremt brukbar til utvikling av kontekstbasert, prosessorientert beskrivelse og forklaring av fenomener.

I dette ligger det at metoden brukes for å utvikle teorier på bakgrunn av de innsamlede data, i motsetning til metoder som baseres på hypoteser som utgangspunkt.

3.4 Kvantitativ versus kvalitativ tilnærming

Det som er skillet mellom kvantitativ og kvalitativ tilnærming beskriver Grønmo (1996) som at begrepsparet i første rekke refererer til en egenskap ved dataene som samles inn. I dette ligger det, at det ikke er metodene som brukes i analysen som nødvendigvis er forskjellige, men en egenskap ved dataene. Det som gjør det hele

litt komplisert er uenighet mellom forskningsmiljøer om hvor grensen mellom hva som kan defineres som kvantitative/kvalitative data skal være.

Grønmo (2004) og Halvorsen (2003) skiller altså på begrepene kvantitativ/kvalitativdata og kvantitativ/kvalitativtilnærming. Hvor situasjonen med begrepsparet knyttes til begrepet data er beskrevet tidligere. Mens begrepet tilnærming bruker de i forbindelse med metodene som brukes for å presentere et resultat. Se omtalen av kvantitativtilnærming og kvalitativtilnærming som er beskrevet tidligere i dette kapitlet.

Selv om mye litteratur omtaler begrepsparet som metoder, betrakter jeg beskrivelsen til Halvorsen (2003) og Grønmo (1996) som klarere, i det de knytter begrepsparet til: Tilnærming. Dette siden metode også brukes om de enkelte forskningsmetoder som brukes innfor begge tilnærminger. Jeg har ovenfor prøvd å være lojal mot forfatternes begrepsanvendelse og dette har medført at begge begreper anvendes. Samtidig har jeg prøvd å bruke begrepet tilnærming der det er snakk om kvalitativ og kvantitativ måte å arbeide på, men der det er snakk om forskningsmetodene innefor måten å arbeide på prøver jeg å bruke begrepet metode. Jeg håper jeg har lykkes med dette skillet.

De betraktninger jeg har funnet som er knyttet til kvalitativ versus kvantitativ tilnærming er i litteratur som omhandler forskning på samfunnet, gjerne til sosiale relasjoner og oppfatninger. En uttalelse som sier litt om hva kvalitativ tilnærming kan bidra med er Kvale (2001): ”Kvalitative metoder er ikke bare en slags ny, myk teknologi som kommer i tillegg til det eksisterende harde kvantitative metodearsenal i samfunnsvitenskapene. Det er heller slik at den kvalitative forskningen innebærer alternative oppfatninger om sosial kunnskap: Om mening, virkelighet og hva som er sant innen samfunnsvitenskapelig forskning.”

Når jeg har lest i litteraturen har det ofte virket som om det er foretatt, eller at du som forsker må foreta et valg mellom kvalitativ/kvantitativ tilnærming. Når denne beskrivelsen knyttes mot metoder som finnes i begge tilnærminger, har jeg opplevd det som at det var ”noe” jeg ikke helt hadde klart for meg. Derfor var det klargjørende å lese drøftingen i artikkelen ”Forholdet mellom kvalitative og kvantitative tilnærminger i samfunnsforskningen ” av Grønmo (1996): Han har et hovedsynspunkt: Kvalitative og kvantitative tilnærminger står ikke i et konkurrerende, men et komplementært forhold til hverandre. I dette ligger det at den ene ikke utelukker den andre innenfor samme undersøkelsesopplegg. Vi kan med andre ord kombinere begge tilnærminger for å få en mer utfyllende anvendelse av datagrunnlaget. Dette er også tema i en artikkel av Tschudi (1996): ”Om nødvendigheten av syntese mellom kvantitative og kvalitative metoder”.

3.5 Mitt valg av metoder

Med bakgrunn i det som jeg har skrevet i kapitlet 3.1 Metodepresentasjon, 3.2 Kvantitativ tilnærming, 3.3 Kvalitativ tilnærming samt 3.4 Kvantitativ versus kvalitativ tilnærming, var min arbeidsform med eksperimentet, klart deduktiv, idet

jeg først laget en hypotese og så skulle prøve å teste den ut (Hoyle et al. 2002). Dette vil jeg prøve å vise i min presentasjon av valg av forskningsmetoder.

3.6 Valg av tilnærming

Min oppgave har i sin natur flere trekk fra samfunnsvitenskapelig forskning, i det jeg ønsket å se på utviklingen av studentenes begrepsforståelse gjennom kurset. Jeg skulle også prøve å registrere deres mening sammenlignet mot "virkeligheten". I det ligger det min oppfattelse av virkeligheten. Dette kommer jeg tilbake til senere.

En likhet med samfunnsvitenskapelig forskning er at jeg ønsket å se på utviklingen av studentenes begrepsforståelse over litt tid. Først under vårt eksperiment og så ved oppfølging senere i semesteret. Det var også viktig å se nærmere på oppfatningen, de selv hadde av situasjonen underveis, siden vi ønsket å vurdere deres opplevelse. Dette passer med: "Et kvalitativt opplegg har som regel til hensikt å få fram hvordan mennesker fortolker og forstår en gitt situasjon." (Jacobsen, 2000). Dette er midt i kjernepunktet for min interesse med prosjektet. Det var å prøve hvordan muntlig kommunikasjon kunne bidra til å bygge begrepsforståelse om objektorientert programmering i et introduksjonskurs.

Dersom vi hadde valgt en kvantitativ tilnærming kunne vi ha brukt spørreundersøkelser, videoopptak med logging av skjerm, eller observasjon for å samle inn data. Uti fra dette materialet kunne jeg laget en kvantitativ analyse. Ettersom vi ønsket å beholde en setting som var mest mulig lik en ordinær undervisningssituasjon, ønsket vi ikke avbrudd for å holde for eksempel spørreundersøkelser. Dette ville eventuelt ha medført at vi måtte ha lagt opp til avbrudd i den faglige del av kurset for å gjennomføre innsamling av data. Derfor ble dette droppet, fordi det ville gi en undervisnings/veiledningssituasjon som var en setting vi mente ikke finnes i ordinære kurs. Nå viste det seg at vi fikk så få deltagere at jeg vil hevde at et resultat av en kvantitativ tilnærming ville jeg ha vanskelig for å forsvare som grunnlag for en oppskalering. Dette kommer jeg tilbake til.

En styrke med kvantitative tilnærminger er at det kan være enkelt å oppskalere resultater til å se virkningen på for eksempel kurs med mange deltagere. For at oppskaleringen skal kunne brukes er det en forutsetning at grunnlagsresultatet kan sies å være generelt gyldig med de angitte parametere. I dette ligger det en tilsvarende gjennomføring og at kursdeltagerne kan sies å være representative for et større antall deltagere. Men uttalelsen til Jacobsen (2000) ovenfor, indikerer derimot at en kvalitativ tilnærming vil være mer egnet for å undersøke hvordan studentene fortolker og forstår situasjonen under kursgjennomføringen. Så min oppgave har en klar kvalitativ tilnærming, selv om jeg enkelte ganger bruker det som tidligere er beskrevet som kvantitative mengdebegreper.

Jeg ønsket å sette studentenes fortolkninger av situasjonen underveis, opp mot hva de uttrykte i diskusjon med hva de gjorde på datamaskinen. Dette å kunne holde data fra flere kilder opp mot hverandre og samtidig få med alternative oppfatninger fra deltagerne er jo nettopp en av styrkene til kvalitativ forskning (Kvale, 2001). Det

at vi da måtte gi slipp på noe av det som er styrken til kvantitativ forskning, nemlig tesen om at data samles inn objektivt og at kvantifisering er en objektiv beskrivelse av en situasjon blir et valg om hva en legger mest vekt på i undersøkelsen.

3.6.1 Valg av perspektiv

Som filosofisk perspektiv ligger oppgaven klart på det interpretive perspektiv. Dette siden vi må forholde oss til de meninger og utsagn som studentene uttrykker. Selv om videoanalysen baseres på en kombinasjon av hva som ble skrevet på tastatur, tegnet på papir eller sagt i gruppene, vil dette ofte sammenlignes med det som ble uttrykt verbalt av studentene utenom opptakene. Siden jeg fokuserer på dialogen og hva den uttrykker, og bruker mer det som skrives på skjermen for å verifisere begrepsforståelsen er dette helt klart et interpretivt perspektiv.

Det finnes element av et positivistisk perspektiv i det vi tok utgangspunkt i at studentenes bakgrunn var som tidligere beskrevet og vi forutsatte at denne beskrivelsen var objektiv. Det kan vel til en viss grad sies at jeg forutsatte at mine observasjoner som bygger på tidligere erfaringer var objektive. At virkeligheten kan objektivt beskrives er i følge teorigjennomgangen tidligere et kriterium for et positivistisk perspektiv. Argumenter om at mine observasjoner var subjektive og hadde for lite grunnlag til å kunne representere en generalisering må jeg akseptere. Det jeg vil fremføre, er at jeg var en person i den gruppen som deltok i arbeidet med å finne en omforent beskrivelse av bakgrunnen til studentene. Samt at jeg ikke fremfører at min beskrivelse er generell, men heller en beskrivelse av hvordan vi som gruppe oppfattet situasjonen.

Det at jeg hadde utgangspunkt i det å fokusere på om vår form for muntlig kommunikasjon kunne gi et bidrag til å bygge kunnskap om objektorienterte begreper i et begynnerkurs, kan sammenlignes med det å teste en teori. Det er jo nettopp det siste som det positivistiske perspektiv brukes til.

Dersom jeg bruker beskrivelsen av kritiske perspektiv som omtalt tidligere, er det lite av et kritisk perspektiv i min oppgave.

3.6.2 Valg av metoder

Jeg skal prøve å vurdere i hvilken grad de enkelte metoder som er omtalt i kapittel 3.3.2 Metoder, kommer til anvendelse i vårt eksperiment.

Aksjonsforskning

Når det foran i kapitlet står at ved aksjonsforskning gjøres det først en undersøkelse, så kan vi kritiseres for at vårt eksperiment primært var basert på gruppens kunnskap om hvorledes innføring av objektorienterte begreper og programmering ble utført på det tidspunktet vi startet prosjektet. Det som jeg i tillegg gjorde, var å lese gjennom

noen bøker som er skrevet for å anvendes i forbindelse med innføringskurs i programmering. Dette kommer jeg tilbake til i kapittel 5 Vurdering av bøker. Nå kan det sies at alle deltakerne i prosjektgruppen hadde jobbet med temaet i større eller mindre grad, i tildels flere år. Så det var nok årsaken til at vi ikke formaliserte undersøkelsesarbeidet mer (1.1 COOL).

Hypotesen jeg jobbet etter, var å prøve ut om en didaktikk som baseres på veiledning og studentdrevet progresjon, kan bidra til begrepsbygging av objektorienterte begreper (1.2 Problemstilling). Dette legger opp til at vi måtte være aktive i vår kommunikasjon med studentene.

Det at vi som holdt kurset, skulle være deltagende ved at vi fortalte og deltok i diskusjonen som en aktiv part, som søkte og bidro med informasjon, er trekk som knytter mot aksjonsforskning som metode. Det vil være mulig å fremføre at vi burde ha hatt et klarere skille mellom kursholdere og observatører. På den annen side er det lettere for kursholderen å observere nært studentene, og stille spørsmål for å avklare tema som en forsker synes ville være interessant å se nærmere på. Denne interaksjonen skal ikke underskattes.

Case-studier

Det er en del momenter i arbeidet som har likhetstrekk med case-studier. Det kan hevdes at vi så nærmere på et tilfelle som finnes i en verden av virkeligheten, idet studentene skulle programmere en robot som kunne treffe på vegger og hente gjenstander. Roboter som agerer slik, finnes det mange av i næringslivet og i leker. Når vi brukte studentenes tidligere erfaringer, så må jo disse nødvendigvis ha kommet gjennom noe som studentene har opplevd før i livet. Dermed finnes det enda en knytning mot det som teorien omtaler som sammenhenger som forekommer i det virkelige liv.

Dette er jo alle momenter som med en viss grad av velvilje kan tolkes slik at min oppgave er en case-studie. Men vi ønsket til en viss grad å kunne kontrollere hendelser og kunne påvirke disse under prosjektet. Dette er noe av det som beskrives som lite egnet for case-studier.

Den største forskjellen mellom aksjonsforskning og case-studier er at i aksjonsforskning er forskeren deltagende og kan påvirke, mens i case-studier har vedkommende kun rolle som observatør.

Etnografi-studier

Det at vi hadde så nær kontakt og samarbeid med studentene kan til en viss grad beskrives som etnografi, men kurset var alt for kort til å kunne defineres som å være innenfor denne type studier. Bruk av parprogrammering og felles diskusjoner anvender jo sosiale virkemidler for å oppnå læring, men som jeg skriver, de er bare et virkemiddel, det er læring av objektorientert programmering som er i fokus. Derfor er heller ikke dette et godt argument for at oppgaven var å betrakte som en etnografiskstudie.

Grounded teori

Siden vi ikke utviklet noe teori på bakgrunn av en undersøkelse er det lite av grounded teori i min oppgave.

3.7 Innsamling av data

Vi måtte på et tidspunkt bestemme oss for hva vi skulle bruke dataene til, og dette må danne grunnlaget for hvilke typer data vi ønsket innsamlet. Alternativ som ble diskutert var: Spørreundersøkelser, videoopptak, logg av skjermer, intervju, observasjon, lese de ferdige program som studentene laget og eksamensresultat. Jeg skal ikke her beskrive alle alternativ i detalj, men bare knytte noen kommentarer til enkelte. Spørreundersøkelser kunne være detaljerte påstander som studentene måtte ta stilling til ved avkryssing eller mulighet for mer utfyllende tekst. Videoopptak gir mulighet for å spole tilbake for verifikasjon, og det igjen kan gi mulighet for å produsere kvantitative data ut i fra hva som kan observeres. Det er også spørsmål om opptakene skal dekke hele eller deler av gruppen, eventuelt med valg av bildeinnhold. Skulle vi for eksempel ha med ansikt eller se skjermen, eventuelt få med notater som ble gjort på papir? Logg av alle skjermer er ressurskrevende når det gjelder lagringsplass. Skulle vi intervjue alle eller et utvalg av studenter?

Et viktig element ved valg av datatype er validiteten av resultatene vi skulle komme frem til. Validitet brukes om hvor gyldig dataene er, det vil si hvor relevant dataene er for problemstillingen (Halvorsen, 2000). Et par andre eksempler på hvordan begrepet kan brukes er intern validitet, som sier noe om vi kan forvente tilsvarende resultat, når vi bruker samme metoder, setting og teknikker, mens ekstern validitet sier noe om hvor generaliserbart resultatet er i en virkelig verden, med andre settinger, deltagere og andre tidspunkt. (Campbell og Stanly, 1963)

Hypotesens to element (1.2 Problemstilling) er en pekepinn om hva vi ønsket oss. Det må her erkjennes at vi under arbeidet brukte en form for iterasjon i arbeidsprosessen for å komme frem til hypotesen og hvilke data vi skulle ha. Styrken i dette er at det øker muligheten for at hypotese og data som samles inn lettere stemmer overens. Dermed kan resultatet av undersøkelsen bli bedre enn den ellers ville ha gjort, siden vi minsker risikoen for at vi legger arbeid i datamengder som det i ettertid viser seg en har liten bruk for, eller at vi senere finner ut at det er noe som vi gjerne skulle ha hatt.

I vårt prosjekt planla vi å benytte maksimalt 20-25 studenter og dette antallet ville nok vært et grunnlag for innsamling av data som egnet seg for kvantifisering. Da spesielt data som kunne samles inn i ettertid, for å se om jeg kunne finne noen signifikant forskjell i begrepsforståelsen for de som hadde deltatt på vårt sommerkurs kontra de andre studentene. Her kunne det for eksempel vært aktuelt å bruke eksamensresultat som datagrunnlag. Vi la opp til at det på et senere tidspunkt skulle være mulig å gjennomføre slike undersøkelser. Nå viste det seg at antall deltakere på sommerkurset ble så liten at vi droppet å gjennomføre slike

undersøkelser. Da med den begrunnelsen at den eksterne validiteten av resultater som innebar en sammenligning mellom våre deltagere og de øvrige som skulle følge innføringskurset ved universitetet ikke ville være av god nok kvalitet.

Vi fant tidlig ut at fokus skulle være å se nærmere på "øyeblikksbilder", det vil si at vi så på hva som var situasjonen der og da, og i mindre grad på hva studentene tenkte på i ettertid. Årsaken til det ligger primært i at vi hadde et ønske om å se på hvordan vårt opplegg kunne bidra til å gi studentene en mestringsfølelse underveis. Dette kommer jeg tilbake til senere.

Det å bruke videoopptak og logging av skjermer er metoder som støtter opp om "øyeblikksbilder" og unngå avbrudd av undervisningen. Slik jeg ser det, er det imidlertid vanskelig å bruke disse metodene for å representere studentenes opplevelse av mestringsfølelse i øyeblikket. Dette kommer jeg tilbake til senere.

Innsamling av data som skulle ligge til grunn for analysen, ble vurdert ut i fra flere innfallsvinkler. Vi kunne basere oss bare på det som ble produsert av studentene ved å logge skjermene som studentene brukte. Dette ville kunne vise med stor presisjon hva som ble produsert, men ville gi oss lite innblikk i prosessen som ligger forut for at de skriver på tastaturet. Den delen av eksperimentet som skulle se på hva studentene klarte å lære av programmering kunne klare seg med en slik form for innsamling. I mitt interesseområde, skulle jeg se på kommunikasjonen og hvorledes denne kunne brukes som bidrag i begrepsforståelsen. Da ville en logg av skjermen være et begrenset bidrag med bakgrunn i det som er skrevet ovenfor, men loggen vil kunne brukes til å verifisere våre observasjoner med det som ble skrevet på tastaturet.

Fra en av forskerne i planleggingsgruppen ble det som tidligere beskrevet, argumentert sterkt for at vi skulle bruke videoopptak av et par grupper, samt at vi skulle prøve å få med på opptaket den delen som instruktørene hadde i plenum. Jeg var redd for at videoopptak kunne påvirke veiledningssituasjonen i så stor grad at settingen kunne oppfattes dit hen at den lignet lite på en veiledningssituasjon som finnes i en ordinær kursavholdelse. Men det ble enighet i gruppen om at vi skulle prøve med videoopptak, med den begrunnelsen at det ville være lettere å se på detaljer i prosessen for å få en bedre dokumentasjon som kunne brukes i vår analyse. Dessuten kunne videoopptak til en viss grad vise hva studentene skrev på skjermen.

Først på tredje dagen erfarte vi at å zoome inn på skjermen, kan gi et bra bilde av innholdet som finnes der. Ved å bruke zoom kunne vi flytte bildeutsnittet ganske greit slik at vi fikk med hva vi i det øyeblikket fant interessant. Vanskeligheten var at vi noen ganger glemte å zoome dersom situasjonen i rommet endret seg. For eksempel at en av instruktørene grep ordet og skulle ta noe i plenum. Jeg ser også av opptakene at dersom bildeutsnittet var på plenum ble det vanskelig å følge aktivitetene i gruppene. Ved et senere eksperiment har jeg erfart at videoopptak kombinert med logging av skjerm og mulighet for parallell avspilling er et godt hjelpemiddel for analysearbeidet.

Vi kunne basere oss på observasjoner ved hjelp av egne notater av vår dialog med studentene. Dette kan gi en indikasjon på utviklingen av begrepsforståelsen underveis. Svakheten er at vi da ikke kan si noe om dialogen internt i gruppene og vi

var av den formening at det var vesentlig for å sjekke om våre observasjoner stemte med det som reelt foregikk i gruppene. Derfor ble videoopptakene så vesentlig for mitt analysearbeid.

Når det gjaldt den muntlige kommunikasjonen vi skulle observere, kan det hevdes at studentenes uttalelse er subjektive uttalelser gjort i en setting som gjør at de kan ha ønske om å uttrykke noe som gir positivt inntrykk på veileder. Ønsket om å gi et slikt inntrykk på veileder kan gjelde selv om vi ikke skulle gi noen formativ evaluering av studentenes arbeid. Studentenes uttalelser kan også bli farget av deres personlige synspunkter på veileder eller andre som var til stede i rommet. Derfor kan det være viktig å benytte flere kilder for å verifisere et resultat. På den annen side er det stor sannsynlighet for at slik farging også kommer dersom samme opplegget brukes i et ordinært begynnerkurs i programmering.

For å vurdere våre egne observasjoner og videoopptak, opp mot studentenes opplevelse, ville vi at studentene skulle gi uttrykke for sin opplevelse og erfaringer underveis. Deres opplevelse kunne vi få gjennom direkte spørsmål som for eksempel: ”Hva oppfatter du at et objekt er? Kan du beskrive hva du har lært i denne aktiviteten?” Vi kunne også prøve å få oppfatninger og synspunkter i diskusjoner under kursets gang. Her gjelder også det som er beskrevet ovenfor om farging av uttalelser.

Dersom vi skulle anonymisert innsamlingen av deler av datagrunnlaget, som for eksempel logging av skjermer eller spørreundersøkelser, ville det blitt vanskelig å sammenligne disse med de data som vi fikk via egne observasjoner. De data som vi fikk gjennom å observere under kurset og de data vi kan få fra videomaterialet er helt klart ikke anonymisert.

Når det gjelder bedømninger i forhold til kvantitative data er dette omtalt tidligere.

3.7.1 Motivasjon hos studentene

Studentenes mestringsfølelse var for meg sentralt. Siden dette kurset var et første møte med programmering skulle vi evaluere om studentene oppfattet kurset som motiverende for videre læring av programmering. Dette med bakgrunn i at vi laget en forutsetning om at vi ville finne ut om opplegget kunne være et bidrag til å redusere det store frafallet av studenter som introduksjonskurset ved Oslo Universitet har.

Jeg hevder at en positiv opplevelse er viktig for avgjørelsen som studenten kontinuerlig tar med hensyn til om vedkommende skal fullføre et kurs. Et vesentlig bidrag til en slik avgjørelse vil være følelsen av mestring slik det er beskrevet i teorikapitlet. Mestring kunne vi ha registeret gjennom for eksempel tester underveis og logging av skjermer. Men slike metoder sier lite om studentens opplevelse av dette. Metoder for å se nærmere på studentenes opplevelse kan være videoopptak, spørreundersøkelser og intervjuer. Til det er det å bemerke at en spørreundersøkelse gjort i etterkant, vil gjøre at studentene lettere kan utnytte modningsprosessen for å gjøre en mer korrekt bedømning av kurset.

Her er det å bemerke at jeg ville registre underveis om veiledningen vi gjorde ble oppfattet positivt med hensyn på motivasjon i et slikt begynnerkurs. Grunnen til at jeg ville gjøre det underveis var at jeg da hadde mulighet til å gjøre situasjonstilpasset korrigeringer på dialogformen. Dette at situasjonstilpassingen ble så sentral i dialogen, kan medføre at undersøkelsen har et preg av etnografi slik det er beskrevet tidligere. En svakhet er at jeg ikke har noe opptak eller observasjoner gjort av en tredje person. I det kommunikasjonen var muntlig har jeg bare notater av mine egne observasjoner, pluss merknader som jeg tok underveis. Den viktigste grunnen til interessen for å registrere en oppfattelse hos studentene med hensyn til kommunikasjon, var nok at jeg hevder at studentens opplevelse der og da er viktigere for motivasjonen under kurset enn hva som studenten kan ha av synspunkter i ettertid.

3.7.2 Begrepsforståelse

En annen sak som vi ønsket å se nærmere på ved hjelp av dialog med studentene, var om den form for dialog som vi hadde i kursgjennomføringen kunne oppleves å gi et positivt bidrag til å bygge forståelse for objektorienterte begrep. Da spesielt ut i fra studentenes ståsted. Det at jeg ville ha en dynamisk relasjon som skulle baseres på dialogen og studentenes erfaring, gjorde at det å bruke metoder som ofte sees i sammenheng med kvalitativ tilnærming ble mest naturlig. Et annet pluss for dette valget, var at vi oppfattet at det da ble en tettere registrering av studentenes opplevelse av dynamikken i utviklingen.

Jeg hevder at med bakgrunn i det jeg har skrevet foran at dette er det som omtales som interpretiv forskning. I tillegg vil jeg omtale det som elementer av aksjonsforskning.

Vi som foresto selve undervisningen, ville svært gjerne ha tilbakemelding fra deltagerne og da gjerne litt i etterpåklokskapens perspektiv. Så det gjennomførte vi ved at vi hadde oppsummeringer ved kursavslutning og ved at vi hadde en uformell kontakt i oktober. På det tidspunktet hadde studentene kommet over midtveis i sitt første ordinære programmeringskurs og kunne uttale seg om sin egen oppfattelse av i hvilken grad sommerkurset hadde hjulpet på begrepsforståelsen i programmeringsarbeidet. Det må sies at vi her bare fikk studentenes egen oppfattelse og ikke hadde mulighet for å etterprøve dette opp mot andre kilder.

For å verifisere vårt opplegg versus et annet burde vi lagt opp til en kontrollgruppe som fulgte et annet opplegg, men vi hadde ikke ressurser til det.

4 Læringsteori

Her vil jeg prøve å presentere noen teorier som ligger til grunn i denne rapporten. Jeg vil presentere noen begreper og prøve å beskrive litt bakgrunn og hvorledes de er å oppfatte og hvordan deres innbyrdes forhold er.

Det å lære noe i en formell institusjon, heretter kaldt undervisningsinstitusjon, baseres ofte på at det anvendes tre elementer: Ved undervisning, egen oppdagelse/ konstruksjon av kunnskap eller i en veiledningssituasjon. Undervisning er en aktivitet som er en preprogrammert sekvens av aktiviteter som ledes av læreren og rettes mot alle studenter, som forelesninger og problemløsning. Egen oppdagelse/konstruksjon av kunnskap gjøres ved å studere litteratur eller jobbe med problemer. Veiledning er derimot interaksjon med studenter basert på deres erfaring, som diskusjoner og "coaching" (Kaasbøll, 2002). Denne definisjonen av undervisning fant jeg i begynnelsen å være for snever i et slikt prosjekt som vårt. Men etter å ha jobbet litt med det, har jeg innsett at definisjonen beskriver at forarbeidet har lagt opp hva som i hovedsak skal skje og i hvilken rekkefølge. Slik jeg tolker det, vil definisjonen av begrepet undervisning også være gjeldende dersom rekkefølgen på aktivitetene endres.

Ordet coaching brukes i dag i stigende grad og flere sammenhenger i Norge. Blant annet i jobbsammenheng og innenfor idretten, for å beskrive spesielle trenerroller. Ordet coaching har jeg lett i mange kilder for å finne et norsk uttrykk som dekker. Det nærmeste jeg finner er kusk, fordi ordet indikerer at det som foregår er å styre studenten mot et mål på samme måte som en kusk styrer hesten. Jeg oppfatter ordet coach som mer drivende med hensyn på fremdrift i aktiviteten, tettere og mer styrende enn veiledning. I dag brukes veiledning mer om det å gi tips og råd, og så er det mer "frivillig" å bruke dem.

Det finnes artikler og bøker som presenterer forskning på temaer knyttet til situasjoner og kunnskap om mennesker i en læresituasjon som kan knyttes til temaet i min oppgave. Jeg skal senere prøve å knytte denne kunnskapen opp mot de observasjoner som ble gjort.

Siden begrepet mestring er såpass sentralt i den didaktikken vi la til grunn, starter jeg med å prøve å avklare litt. Det er verdt å merke seg at det er vesentlig forskjell på det som i litteraturen omtales som mesterlære og uttrykket, det å mestre noe. Først beskriver jeg begrepet mesterlære for så å ta for meg begrepet mestring.

4.1 Mesterlære

Boken "Mesterlære, læring som sosial praksis" (Nielsen og Kvale, 1999) beskriver at uttrykket mesterlære kommer fra en undervisningsform, som har blitt brukt i århundre som en formalisert undervisningsform. Da er det ofte knyttet til at unge mennesker blir opplært i ferdigheter, kunnskaper og verdier som knytter seg til et

håndverk og et yrke. Formen består i det vesentlige av å følge en mester og lære via å se på mesteren utføre arbeid, etter hvert får lærlingen selv prøve seg under veiledning av en mester. Denne formen for undervisning har på mange områder blitt erstattet av formell undervisning knyttet til en skole.

Boken beskriver flere eksempler på mesterlære og aktualiteten av bruken. Boken refererer blant annet et legetidskrift "..mesterlære som fremtidens utdanningsform:", "Mesterlære med spesialisten i front." Men boken beskriver også at denne type utdanning gis svært liten omtale innenfor fagområdene pedagogikk og psykologi. Det at den har liten omtale innenfor de omtalte fagområdene kan komme av at den i hovedsak brukes i yrkesrelaterte utdanninger og ofte knyttes til fagbrev. Dermed får den ikke samme fokus som tradisjonell undervisning i skoleverket og i akademiske miljø. Men som eksemplene viser, anvendes den også innenfor mer akademisk miljøer.

Et eksempel på mesterlæring som jeg har fått beskrevet, var under arbeid med et kurs på universitetet i Oslo. Under et intervju med en lege som var spesialistutdannet, brukte han et eksempel på hvordan kunnskap gjør at en kan se noe som ikke andre ser. Når det utdannes spesialister på å analysere røntgenbilder, så plasseres legene i starten av utdanningen foran en lysvegg hvor det henger noen røntgenbilder og de får studere disse. Så kommer en spesialist som kan fortelle mye som legene ikke ser. Slik fortsetter det gjennom spesialistutdanningen og etter hvert ser legene mer og mer. Han beskrev dette som en måte å bygge kunnskap, hvor en etter hvert kan oppdage informasjon som ikke er synlig eller som en ikke er bevisst på eksisterer før en har kunnskapen.

Jeg sammenlignet det med å lære å gå og han bekreftet dette. Men gjorde oppmerksom på at det kreves også en viss innsikt i prinsippene. Når vi lærer å gå, anvender vi mye kunnskap som en ikke er bevisst på. For eksempel, hvilke muskler vi anvender for å løfte og bevege foten, samt at vi må ha en vektforskyvning fra den ene foten til den andre før vi løfter foten fra bakken. Dette er kunnskap som vi lærer oss ved å se på andre mennesker som går, men kunnskapen er ikke bevisst. Men selve det å gå krever en viss forståelse av prinsippene og det er det som gjør at noen autister må lære disse for å klare å gå.. Denne kombinasjonen av ubevisst og prinsippforståelse gjør at det ikke er slik at vi kan be en vilkårlig person om å beskrive alle elementer som er nødvendig for å gå. Disse kombinasjonene er i overensstemmelse med uttalelsene i boken til Nielsen og Kvale (1999).

Denne formen å bygge kunnskap på, ved å lære noe som ikke nødvendigvis behøver å være basert på at alle elementer må være bevisst og at vedkommende kan beskrive alle disse elementene brukes i profesjonsutdanningene. Denne type utdanning beskrives senere.

Det er kjent at det kan ligge data som kan brukes til informasjon i et bilde, og at informasjon kan være utilgjengelig for oss, før vi prosesserer bildet ved en digital image prosessering ved hjelp av datamaskin. Eksempler på dette er moderne røntgenteknologi og bildeanalyse hvor en kan få se data som i utgangspunktet ikke var synlig på et bilde. Dette er beskrevet nærmere i "Digital Image Processing" (Gonzalez og Woods, 2002). Selv med slike hjelpemidler trengs det fagkunnskap, som kan transformere disse data til informasjon som kan anvendes.

Mesterlære kan utnytte dette at vi kan lære enkelte ferdigheter, som er anvendelig uten at vi nødvendigvis behøver å være bevisst alle teorier og elementer som brukes. Dette betyr ikke at alt kan læres via mesterlære. Eksempel på noe som er mindre egnet for mesterlæring er teoretisk kunnskap som krever forståelse av sammenhengen mellom teorier, som for eksempel matematikk.

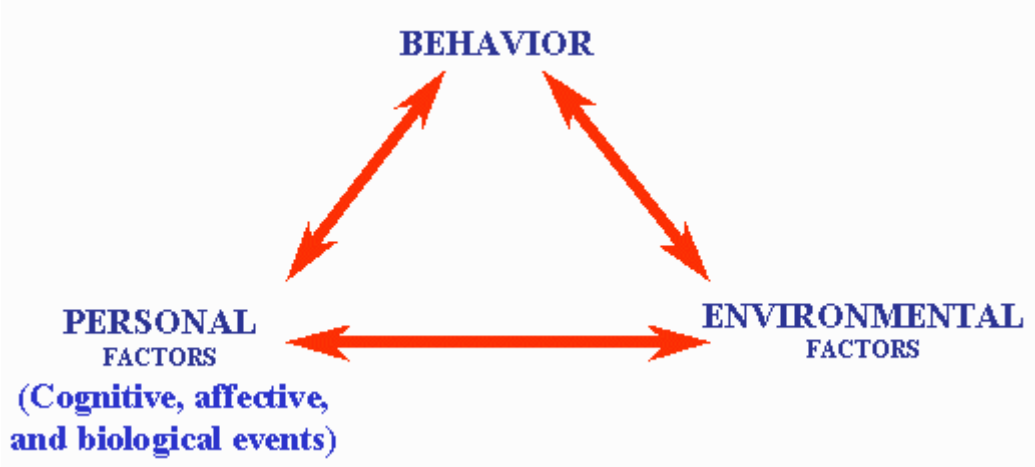
4.2 Mestre

Uttrykket å mestre brukes på norsk om blant annet, det å beherske, greie, det vil si mestre en oppgave, en teknikk (Bokmålsordboka).

4.2.1 Beskrivelse av begrepet

Jeg har fra min trenerutdanning innenfor idrett vært borti uttrykket å mestre. Da har det blitt brukt om at det å mestre er en viktig faktor for motivasjon for utøvere. Jeg har selv prøvd å bruke delmål som utøverne skulle mestre som innlæring og motivasjon for å lære mer kompliserte samhandlinger innenfor en lagidrett som ishockey. I trenerarbeidet brukes det å oppleve graden av mestring blant annet som et virkemiddel for å gi utøverne en klarere innsikt i sine styrker og svakheter. Når Olympiatoppen holder kurser og foredrag brukes ordet som beskrivelse av noe som er vesentlig for å motivere for treningsarbeidet, både for barn og for eliteutøvere. Faktisk sier Bandura (1986) at hvordan folk utfører noe kan ofte bli bedre forutsagt av hva de tror om sin kapasitet på hva de kan mestre enn av hva de faktisk er kapabel til å utføre. Via forskningspersonal ved Norges Idrettshøgskole ble jeg oppmerksom på at på engelsk omtales mestring som self-efficacy. Self-efficacy er beskrevet av Bandura (1986) i hans sosialkognitive teori slik: ”perceived self-efficacy is a significant determinant of performance that operates partially independently of underlying skills”, og “People’s beliefs about their capabilities to produce designated levels of performance that exercise influence over events that affect their lives.” (Bandura, 1998)

Pajares, (2002) plasserer uttrykket mestring (self-efficacy) inn i sosialkognitiv teori ved hjelp av en konseptuel modell som viser avhengighetene:



Figur 3: Overview of social cognitive theory (Pajares, 2002)

Innefor forskning på sykepleie og undervisning ser en på mestring som et moment. Dette kommer frem i Glanz et al, (2002) "Health Behavior and Health Education. Theory, Research and Practice." Her beskrives mestring slik: "*Self-efficacy*: The person's confidence in performing a particular behavior; Approach behavioral change in small steps to ensure success." Det er verdt å merke seg at en ser på små steg som viktig for å få suksess, og i forbindelse med å utvikle ferdigheter i det å sette sprøyte står det: " Simplifying each step and allowing individuals to practice each step in isolation with many repetitions enables them to build **self-efficacy** about performing each step. When persons are **self**-confident about each step, they can progressively put the steps together and build…"

4.2.2 Sykepleier/programmererutdanning

Det kan hevdes at sykepleierutdanning er noe helt annet enn det å lære seg å programmere, men begge utdanningene har en del fellestrekk. Idet de begge er profesjonsutdanninger hvor det å lære ferdigheter er sentralt. Ferdighetene skal brukes til å påvirke noe som bare indirekte kan observeres. Med indirekte mener jeg her at sykepleierstudenten ikke direkte kan se hva som skjer i kroppen. Det kan observeres via teknologisk utstyr, pasienten forteller eller ved å observere pasientens ytre. Studenten som lærer programmering kan heller ikke se hva som skjer i maskinen, vedkommende observerer på en skjerm eller noe som datamaskinen styrer.

Har de to profesjonene fellestrekk i sitt arbeid med det ikke observerbare? De hendelser som skjer og som ikke er observerbar resulterer i en eller flere hendelser som kan observeres. Det å tolke observasjonene og de direkte årsaker kan være nokså trivielt. Eksempler på dette er diare, som kan knyttes til problem i fordøyelsessystemet og et skjermbilde som viser feil format som kan knyttes til problem med den informasjonen som sendes til skjermen. Observatøren bruker sin kunnskap om fysikkens oppbygging og prosesser som er skjult innvendig samt hvorledes de kan påvirkes med medisin, kosthold eller programkode for å forbedre situasjonen.

4.2.3 Motivasjonsfaktor

Hva som utvikler opplevelse av mestring hos folk kommer i hovedsak fra fire hovedkilder. Det mest effektive er å bygge en sterk følelse av å beherske gjennom vellykkede eksperimenter. Suksess bygger en robust tro på ens evne til å mestre noe. Fiasko underminerer den, spesielt hvis fiasko skjer før mestringsfølelsen er blitt litt etablert. Den andre måten å kreere eller styrke mestringsfølelsen er gjennom å observere rollemodeller. Det å observere disse, styrker egen tro på at en har kapasitet til å mestre tilsvarende aktiviteter. Det tredje er overtalelse fra andre, dette styrker folks tro på at de har det som skal til for å få suksess. Dette fordi de mobiliserer mer innsats og utholdenhet når problemer oppstår. Den fjerde måten å styrke mestringsfølelsen på er å redusere folks stressreaksjoner og negative opplevelser (Bandura, 1998).

Dette indikerer at mestring er vesentlig for å motivere for innlæring av ferdigheter som grunnleggende programmering. Dette blir bekreftet i en artikkel av Pajares and Schunk (2001). Her står det: ”High self-efficacy helps create feelings of serenity in approaching difficult tasks and activities. Conversely, people who doubt their capabilities may believe that things are tougher than they really are, a belief that fosters stress, depression, and a narrow vision of how best to solve a problem. Not surprisingly, confidence in ones’ academic capability is a critical component of school success. “

Ut i fra dette kan en hevde at det å skape en følelse hos studentene av å mestre vil gi suksess med læring av programmeringsferdigheter. På den annen side kan en si at det å lykkes med noe skaper positiv selvoppfattelse. Da blir spørsmålet om selvoppfattelsen er årsak eller effekt til akademiske prestasjoner. Pajares og Schunk (2001) skrev at dersom utgangspunket til forskere er å fokusere på en persons selvoppfattelse, hevdes det at selvoppfattelsen er primær årsaken til en students prestasjoner. Mens forskere som har fokus på utvikling av ferdigheter hevder at selvoppfattelsen er en konsekvens av de akademiske prestasjoner. En annen og enklere variant er: Vi gjør det bra på grunn av at vi føler oss vel og vi føler oss bra fordi vi gjør det bra.

Hittil har argumentasjonen vært basert på psykologistudier og studier gjort i akademiske miljø. Jeg vil nå vise at det også finnes arbeider som knytter dette mer spesifikt mot aktiviteter som kan ligne på det å lære grunnleggende programmering. I studier av college studenter som tar vitenskapelige og ingeniørkurs, har det blitt påvist at høy mestring influerer på den akademiske utholdenhet som er nødvendig for å få høy akademisk prestasjon. Dette skriver Pajares og Schunk (2001) med refereranse til: (Lent, Brown, & Larkin, 1984, 1986; and see Hackett, 1995, and Lent & Hackett 1987, for reviews of the influence of self- efficacy on career choices and decisions.)

4.3 Profesjonsutdanning

Programmering inngår i flere profesjonsutdanninger ”profesjon: (fra lat. 'offentlig angivelse av erverv') fag, yrke, levevei, være sanger av p- | gjøre p- av noe drive som erverv ” (Bokmålsordboka). Typiske utdanninger er for eksempel, flere ingeniørretninger, utdanninger i informasjonsteknologi og i en del lærerutdanninger. Profesjonsutdanninger har kanskje med bakgrunn i at de retter seg mot en yrkeskarriere innenfor den enkelte profesjon, lettere enn andre studier for å endre didaktikken ut i fra hva undervisningsinstitusjoner ser som tjenelig for yrkesutøvingen. Jeg har ikke funnet noe dokumentert arbeid, som spesifikt tar for seg en sammenligning mellom profesjonsstudier og andre studier, med hensyn på tilpassingsevne i undervisningsopplegg relatert til en yrkeskarriere.

I følge Jensen (1999) er det ”..gjort et omfattende reformarbeid innenfor enkelte profesjonsutdanninger. Eksempel på dette er medisinsk fakultet på Universitetet i Oslo der det er utviklet en studiemodell med utgangspunkt i problembasert læring. Endringene i synet på kunnskap avspeiler seg også i en fornyet interesse for praksisdelen innenfor profesjonsutdanningene”. Dette viser at begrepet profesjonsutdanning ikke nødvendigvis bare knyttes til det å lære noe praktisk. Men det indikerer at måten en lærer praktisk utøvelse på endrer seg. I dette ligger det at en går mer bort fra en teoretisk overføring av kunnskap til at kunnskapsbyggingen mer skjer gjennom at studenten selv erfarer.

Det å lære programmering er en profesjonsutdanning hvor studentene lærer seg ferdigheter i å skrive program på en form som datamaskiner kan tolke og oversette til et maskinspråk. Denne ferdigheten har i flere bøker fokus på å lære syntaksen i programmeringsspråk. (“*Syntax* studies the rules for combining words into legal phrases and sentences, and the use of those rules to parse and generate sentences.”(Luger, 2005). Mens å bygge kunnskap om begreper og konsekvensen av bruken i programmering, er mer å fokusere på semantikken i programsetningene. *“Semantics* considers the meaning of words, phrases, and sentences and the ways in which meaning is conveyed in natural language expressions.” (Luger, 2005).

4.4 Problembasert læring

4.4.1 Beskrivelse av begrepet

Problembasert læring (PBL) bruker en didaktikk basert på at studentene skal bygge kunnskap gjennom erfaring. I dette ligger det at fokus i undervisningsrommet skifter fra undervisning til læring. Den sentrale forutsetningen er at de fleste studenter lettere vil lære informasjon og ferdigheter hvis de har behov for dem. Behov oppstår når studenter prøver å løse spesifikke, åpne problemer (Burch, 2001). Dette er jo samme strategi som vi valgte i vårt sommerkurs og jeg vil beskrive denne didaktikken litt nærmere. Tankegangen i PBL er godt illustrert av Duch på hjemmesidene til University of Delaware:

"How can I get my students to think?" is a question asked by many faculty, regardless of their disciplines. Problem-based learning (PBL) is an instructional method that challenges students to "learn to learn," working cooperatively in groups to seek solutions to real world problems. These problems are used to engage students' curiosity and initiate learning the subject matter. PBL prepares students to think critically and analytically, and to find and use appropriate learning resources".

Problembasert læring begynte ved McMaster University Medical School i Canada ved at undervisningsmetoden ble introdusert i 60 årene.(Woods, 2005) Siden har det blitt implementert i flere høyere utdanninger rundt om i verden. Den er også adoptert inn i utdanninger på lavere nivå.

Bereiter, og Scardamalia, (in press) skriver om PBL: Problembasert læring blir ofte oppfattet som synonymt med prosjektbasert læring. Men Problembasert læring kommer fra en annen tradisjon med et annet fokus. PBL var designet for å lære medisinsk kunnskap og ferdigheter ved: "…engaging students in solving problems similar to those they could expect to encounter in practice—that is, problems of diagnosis and treatment involving simulated cases." Det forventes at studentene seg i mellom avgjør hvilken informasjon de trenger for å løse et problem, skaffe og dele informasjon og jobbe frem mot en løsning med veiledning, men lite direkte hjelp av instruktør.

4.4.2 Bruk i utdanning

PBL utdanningssystemet omtales ofte som mer ressurskrevende enn tradisjonell forelesning, siden det krever mer veiledning. En av grunnene til at PBL kan være mer kostbart er at opplegget baseres på at en veileder (lærer) skal være tilgjengelig og at studentgruppene ikke bør være for store. Det finnes to sentrale prinsipp i PBL: Åpne problem og studentgrupper som jobber sammen. Problemer er redskapet for læring og grupper er drivstoffet. Problemene transporterer studentene fra en klasseromsverden til en reell situasjon i den virkelige verden, som stimulerer deres vitebegjærlighet og kreativitet. Det andre prinsippet er at læringen skjer i grupper som undersøker problem med å koordinere sin innsats mot et felles mål og ved hjelp av samarbeid presenterer et resultat.(Burch, 2001).

Grunnen til at grupper er så sentrale i denne typen undervisningsopplegg, tar utgangspunkt i tre delmål.

- Ferdighet i problemløsning
- ferdighet i å tilegne seg kunnskaper på egenhånd
- ferdighet i gruppesamarbeid.

Ferdigheter i problemløsning er sentralt når en i en yrkeskarriere skal delta i utvikling av datasystemer eller i andre oppgaver. I undervisningssammenheng er problemløsning enten et hovedmål eller et middel. I enkelte kurs er fokus på det å løse et problem, mens i andre kurs er problemløsningen et middel for å lære noe. Et

eksempel på det siste kan være et programmeringskurs hvor fokus er å lære syntaks i programmeringsspråk.

Det kan skje at studentens fokus blir annerledes enn målet for kurset skulle tilsi. Årsaken til dette kan være en svakhet i opplegget slik at studenten opplever at det legges opp til at problemløsning er det sentrale mål. På den annen side vil det å mestre det å presentere en løsning gi en god følelse som kan motivere for nye utfordringer. Derfor vil måten problemet presenteres på, og veiledningen underveis være vesentlig for at studenten har fokus på det som er målet for kurset.

Som det fremgår ovenfor baserer i utgangspunktet PBL seg på små studentgrupper. Det jeg har funnet omtalt av undervisningsopplegg på universiteter og colleges har ofte 3-8, ja opptil 12 studenter i hver gruppe. Når det gjelder klassestørrelse er det som oftest klasser på rundt 30 stykker. Et av unntakene er omtalt i boken ” The power of problem-based learning” (redigert av Duch, Groh og Allen, 2001). Her omtales klasser med opptil 480 studenter. Det beskrives flere strategier for å bruke PBL for småklasser i store klasser:

- Skaffe flere personer til å delta i oppfølging av gruppene
- mer struktur, slik at det minsker sjansen for at grupper kommer på feil spor
- bruke miniforelesninger for å minske behovet for tett oppfølging av gruppene
- opplæring i PBL i små klasser, før en starter med store klasser.

Teksten viser til en sammenligningsundersøkelse mellom en klasse på 120 og en klasse på 240 studenter. Her viste det seg at klassen på 120 fikk bedre resultater med hensyn på læring enn den større klassen. I den forbindelsen fremfører de at årsaken kan være at den enkelte student fikk mer oppmerksomhet i den minste klassen. Det på grunn av tettheten med studenter i rommet med 240, gjorde det umulig å nå alle studentgrupper.

Jeg har funnet dokumentasjon på at eksamensresultater over tid er de samme for PBL og undervisnings opplegg basert på forelesninger. Et eksempel på det er et introduksjonskurs i OOP (Greening, et al. 1996). Videre har jeg funnet eksempel på en rapport som registrerer forskjell i opparbeidet kompetanse i programmering ved å benytte de to metoder. Rapporten beskriver en sammenligning av to forskjellige kurs, et innføringskurs i programmering og et innføringskurs i computer science (Barg, et al. 2000). Begge kursene er observert over tre år, og kursene er lagt om fra en forsiktig start med PBL i parallell med det tradisjonelle kursopplegget, til full omlegging til PBL. Antall studenter per semester var mellom 500 til 900. Derfor er det interessant at dokumentet er så klar i sine konklusjoner, blant annet sier de: ”.. substantial improvement in basic programming competence”. Jeg må gjøre oppmerksom på at rapporten også fremfører at overgangen til verktøyet Blue (en forløper til BlueJ) har bidratt til forbedring av resultatene. Jeg har i mine søk i literattur ikke funnet noe dokumentasjon på kurs innefor domenet programmering av datamaskiner, som har svakere resultat for PBL baserte kurs sammenlignet med tradisjonelle kurs.

PBL hjelper studentene til å erverve de ferdigheter de trenger for å forsette utdanning på egen hånd, samt at dette er ferdigheter som kommer til anvendelse i

mange andre sammenhenger i livet. Overgangen fra et tradisjonelt undervisningsopplegg, som er basert på at forelesning er et sentralt element, til et system som baseres på at studenten først må finne ut hva vedkommende skal lære og så skaffe den nødvendige informasjonen, kan for en del studenter bli vanskelig.

Men det å skaffe seg kunnskap på egen hånd kan komme i konflikt med interessen til gruppen. Det å sette mål som skal føre frem mot et løsningsforslag, kan kanskje gå greit siden alle i gruppen har interesse av at resultatet blir bra. Et eksempel på konflikt kan være at individene i gruppen kan ha egeninteresse av å få bedre kunnskapsuttelling enn de andre. Dette kan ytterligere forsterkes dersom det for eksempel skal være en eller annen fordeling av karakterer i kurset. Da kan det være et bidrag med uttrykket som fotballtrener Eggen bruker for å motivere spillerne til å bidra til et felles mål: Du blir bedre av å bidra til at de andre blir bedre.

Derfor blir ferdigheter i gruppesamarbeid vesentlig for et godt resultat.

4.5 Parprogrammering

Parprogrammering er en teknikk som brukes i undervisningssituasjoner, men også i større utviklingsprosjekter i næringslivet. Sommerville (2004) beskriver parprogrammering slik: ”Developers work in pairs, checking each other’s work and providing the support to always do a good job.”

Interessen rundt parprogrammering har i de siste seks årene vært stor, mens det har vært utført lite forskningsarbeid på arbeid utført i industrien, finnes en økende mengde forskning utført på studenter som jobber med parprogrammering (Herzog, 2005).

I sin Cand. Scient oppgave henviser han til forskningsresultater som viser at programkvaliteten øker ved bruk av metoden. Det er et fellestrekk at flertallet av studentene trives med metoden og fortrekker parprogrammering. Når det gjelder læring henviser han til positiv effekt på prosentandelen som fullfører kurset. Det ser også ut til å være en positiv effekt på karakterresultatene selv om han bemerker at disse ikke er entydig. Resultatene fra hans eget eksperiment er i overensstemmelse med de forskningsresultatene han henviser til.

Brain Hanks har utført flere forsøk med parprogrammering og i forbindelse med 10th annual SIGCSE conference on Innovation and technology in computer science education ITiCSE '05 har han en artikkel som beskriver et forsøk gjort under et begynnerkurs i programmering og kan vise til ″Pair programming has been shown to provide many pedagogical benefits, particularly for students who are learning to program” (Hanks, 2005).

Det å jobbe i par bidrar i følge Williams og Robert (2000) til at deltagerne holder hverandre kontinuerlig fokusert og at det jobbes med aktiviteten. Det kan også bidra til at diskusjonene avdekker og klargjør forkskjellige oppfattninger av begrepene og hvorledes de kommer til anvendelse i programmeringen. En fare med å jobbe i par, kan være at den ene blir så dominant at den andre bare er til stede uten aktivt å ta del

i arbeidet. Og siden deltagelse er så viktig for læreprosessen bør det være slik at arbeidsoppgavene er utformet slik at de fremmer aktivitet for begge. Veiledning fra instruktører må også prøve å fange opp og korrigere dersom gruppene ikke fungerer som planlagt.

4.6 Veiledning baseres på tre spørsmål

Det finnes flere måter å legge opp veiledningen på, her skal jeg beskrive en metode som jeg ble kjent med da jeg leste en artikkel av Schoenfeld (1992) om undervisning i problemløsning i matematikk. Jeg har funnet mange artikler som siterer denne artikkelen. Jeg har lest noen, og samtlige drøfter prosjekter som fokuserer på undervisningsmetoder i matematikk. Jeg har ikke funnet noen som omhandler temaet hans innen domenet for å lære programmering. Schoenfeld fokuserer på å bygge innsikt i temaer. Han skiller klart mellom teknikker for problemløsning som kan anvendes ved rutineøvelser, og arbeid med virkelig problemløsning på ukjente problemer. Arbeid med rutine øvelser beskriver han som problemløsning hvor problemkonteksten forteller studenten hvilken teknikk han skal anvende (for eksempel hvilken matematisk formel), mens det han omtaler som virkelig problemløsning er problemløsning hvor problemkonteksten ikke forteller studenten hvilken teknikk vedkommende skal velge.

Relatert til denne oppgaven kan det sies at i arbeidet med aktivitetene var studentene kjent med hvilket begrep som var tema, og at det derfor var å sammenligne med det han beskriver som en rutineøvelse. Men jeg vil betrakte våre aktiviteter som vi presenterer for studenten til å være utenfor det Schoefeld beskriver som en rutineøvelse. Våre aktiviteter er for studenten å betrakte som et virkelig problem, idet objektet skal bevege seg og programkoden skal håndtere hindringer som ligger der. Dette fordi selv om begrepsforståelsen er diskutert i plenum, er selve problemet som skal løses slik det for eksempel er beskrevet i Figur 1: Eksempel på aktivitet, ukjent for studenten. Jeg gjør oppmerksom på at når studentene jobber med denne aktiviteten, har de ikke fått presentert noen skisse til løsning eller sett kode som bruker en stoppbetingelse.

Schoenfelds resultater er basert på det som omtales som problemløsningsklasser. Det er klasser med grupper som består av tre eller fire personer, så går læreren og bedriver coaching (4 Læringsteori) mens studentene jobber med problemer. For å stimulere til at studentene jobbet med det de måtte oppleve som problem eller vanskelig fremfor å søke lettvinte løsninger, ønsker forfatteren å stimulere studentenes egen kritiske tenking. Dette gjør han gjennom å unngå å fortelle løsningen eller selv hinte om den. Han har reservert seg retten til å kunne stille tre spørsmål når som helst:

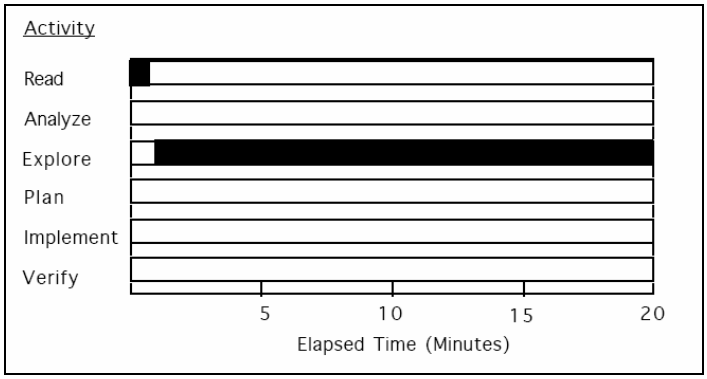
- "What (exactly) are you doing? (Can you describe it precisely?)
- Why are you doing it? (How does it fit into the solution?)
- How does it help you? (What will you do with the outcome when you obtain it?)"

  (Schoenfeld, 1992)

Som en ser av teksten leder den studentene til stadig å prøve og evaluere produktet vedkommende jobber med opp mot hva som er problemet. Anvendt i programmeringsdomenet oppfatter jeg det slik at spørsmålene skal lede studenten til å tenke gjennom hva koden utfører eller hva som er hensikten med koden. Når studenten tvinges til å verbalisere sine tanker, kan dette også bidra til at studenten selv må tenke gjennom hva koden representerer. Det kan hevdes at enhver diskusjon om et kodeeksempel vil tjene samme nytte. Jeg hevder at nettopp spørsmålenes utforming gjør at studentene avklarer tidligere om kodeforslaget gjør det som de ønsker og om det kan bidra positivt i en løsning. Som for eksempel når de har brukt en while setning og egentlig trenger en if setning. Dette kommer jeg tilbake til i drøftingen i kapittel 9.3.1 Situasjonsbetinget og de tre spørsmålene. Jeg hevder også at det er slik at en diskusjon i gruppen kan bidra til å avklare synspunkter på forslaget. Dette at studentene bruker diskusjon og argumentasjon i læringsprosessen er et av hovedargumentene for problembasert læring (4.4 Problembasert læring).

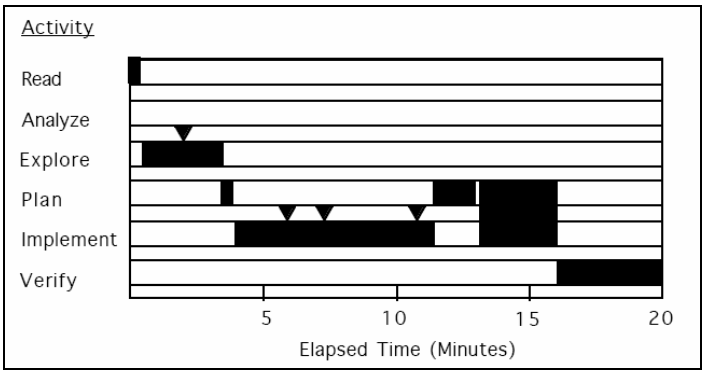
Forfatteren tar spørsmålene i bruk allerede i starten på sine kurs. Forfatteren er meget klar på at det er et viktig poeng at studenten er kjent med at det er disse spørsmål som vil bli stilt i en veiledningssituasjon. Dette for at settingen ikke skal oppleves som en overraskelse og at de skal være kjent med at spørsmålene skal stimulere til kritisk tenking og lede dem til å lage et løsningsforslag. Ofte klarer ikke studentene i starten å svare på dem, men han forsetter å bruke dem. Dette leder til at studentene begynner å forsvare seg mot dem ved å forberede svarene ved å diskutere seg i mellom mulige svar før de stiller spørsmålene. Før kurset er ferdig vil han at denne diskusjonen skal bli en vane (Han gjør et poeng av at det må påregnes at det kan gå over halve semesteret før det skjer).

Resultatene hans viser en klar forskjell på studentene før kurset og etter kurset når det gjelder teknikker for å løse problem som for studenten er ukjent på forhånd. Han bruker en figur for å illusterere en typisk arbeidsform for en student før kurset:



Figur 4: Typisk arbeidsprosess før kurset.

Figuren illusterer at den typiske student leser oppgaven, velger en angrepsmåte og holder seg til denne selv om det finnes klare tegn på at det ikke fører frem. Derimot er det slik at studenter som har gjennomgått kurset får en annen profil på arbeidsprosessen. Her bruker jeg en figur som illusterer prosessen til to studenter som etter kurset jobber med å løse et problem.



Figur 5: To studenters arbeidsprosess etter kurset.

Ser vi på denne figuren så viser den at studentene prøver flere løsninger, men da den enkelte ikke fører frem prøver de neste, til de finner en løsning som fører frem. Det er nettopp de tre spørsmål som bidrar til å bygge kunnskap hos studenten om å evaluere forslag de jobber med opp mot målet. De finner derfor raskere enn den typiske student (Figur 4: Typisk arbeidsprosess før kurset) ut om forslaget kan brukes. Jeg hevder også at slik erfaring gjør at de lettere tør prøve et forslag.

Det kan hevdes at mitt mål var urealistisk ut i fra Schoenfelds uttallelse om at det kunne gå over halve semestret før studentene gjorde det til en vane å forberede spørsmålene ved en diskusjon i gruppen. Slik jeg oppfatter hans artikkel er de problem som studentene i vårt kurs stilles ovenfor mer begrenset, idet studentene vet at det bare er ett nytt problem som skal løses i hver aktivitet.

Jeg ønsket primært at vi skulle fokusere på å bruke de tre spørsmål opp mot anvendelsen av de objektorienterte begrepene og i mindre grad på syntaks i programsetningene.

4.7 Programmerings-paradigmer

Når jeg ser på noen lærebøker (5 Vurdering av bøker) og undervisningsopplegg som jeg har funnet på nettet, finner jeg lite orientering om de forskjellige paradigmer. Selv det paradigme som er valgt som grunnlag i et kurs, er sjelden tydeliggjort og referert til som et grunnleggende prinsipp for elementer i opplegget. I det ligger det at jeg ikke finner noe mer enn kort stadfesting eller kommentar i dokumentasjonen.

Jeg beskriver litt om de forskjellige paradigmer og knytter dette til undervisningssituasjonen. En som har beskrevet dette er Bergin (2000) og jeg bruker han som kilde.

Om programmeringskurs har det blitt sagt "…teach object languages and systems. Doing so requires understanding of some underlying principles that greatly affect the pedagogy we must use." (Bergin, 2000). Jeg skal anvende meg av tre av de underliggende prinsipper for å beskrive det prosedurale og objektorienterte paradigme.

Det prosedurale paradigme fokuserer først og fremst på algoritmer, og så lages det en datastruktur som passer de prosesser som er laget. Dette fungerer noen ganger, men senere vil jeg vise at OOP (objekt orientert programmering) snur dette fokus rundt.

En rimelig typisk komponent i et begynnerkurs i programmering som bruker den prosedurale paradigme, er å anvende von Neumann maskinarkitektur for å beskrive for studentene hva som skjer under utførelsen av et program. Den legger opp til at RAM i maskinen inneholder programmer (prosedyrer) og data. Disse sendes til CPU'en for å bli prosessert. Programmet utføres ved at data transformeres fra et initsielt nivå til et endelig resultat.

Denne modellen stemmer godt overens med det prosedurale paradigmet. Men den stemmer mindre bra overens med den funksjonelle og det objektorienterte paradigmet.
Det funksjonelle paradigmet skjuler selvsagt den fysiske arkitekturen. Det objektorienterte paradigmet skjuler ikke den fysiske arkitekturen, men isteden for at data blir flyttet til CPU for å prosesseres er det en vanlig metafor, at CPU beveger seg til objektet for å gi objektet prosesseringskraft. Når vi undersøker OOP (objekt orientert programmering) videre, ser vi igjen og igjen at OOP ikke er en utvidelse av prosedural paradigme, men heller snur tankeprosessene på hodet.

Siden prosjektet hadde fokus på objektorientert programmering skriver jeg litt utførlig om dette og jeg bruker fortsatt Bergin (2000) som kilde.

Objektorientert programmering innbærer at objekter innkapsler algoritmene, så vel som data. Programmet selv kan sees på som elementer som har en interaksjon med hverandre. Hvert av disse elementene kan fungere som en leverandør av data eller tjenester til andre elementer.

En anerkjent metode for å utvikle et objektorientert program, er å finne frem til objektene via en prosess av simulering. Objektene i systemet skal representere objekter i en virkelig verden. Dette gjør at utviklerne ser etter objekter å modulere, ikke etter prosedyrer. Disse objektene skal ha en handling (metoder i Java), heller enn at de er objekter som det utføres en handling på. Med handling menes her en programmodul som utfører endring av data i eget objekt eller leverer en tjeneste til et annet objekt. Dersom noe ikke har en handling er det neppe et objekt.

En oppsummering er at et proseduralt program er som et tre av funksjoner og et objekt program er som en web av klienter og tjenere. Dette innbærer at verden er nokså forskjellig i de to paradigmer.

Erfaring fra industrien er at en erfaren prosedural programmerer vil bruke et år til 18 måneder, for å klare overgangen (Stroustrup, 1994). Lattanzi og Henry (1996) rapporterer også om vanskeligheten med å lære objektorientert programmering til studenter som har erfaring i prosedural paradigme. Når programmererne er i lærefasen vil de naturlig prøve å løse problemer ved å dekomponere funksjoner, heller enn å prøve å oppdage objekter. Det tar tid å endre til en ny måte å tenke på.

Bergin (2000) argumenterer også for at det er lettere å gå fra OO paradigmet til prosedural enn den andre veien. Dette fordi en lærer om funksjoner i OOP. Gode prosedurale program kan bli laget av OOP programmerere fordi de vil helt naturlig partisjonere sine program inn i objektlignende enheter av funksjonalitet selv om de ikke har samme vertøyet som de er vant med fra OOP verden.

Dette er et klart argument for rekkefølgen ved utvikling av undervisningsopplegg bør være: Objektorientert først og så eventuelt lære litt om de andre paradigmene når en skal lære å programmere.

5 Vurdering av bøker

I dette kapitlet ser jeg nærmere på noen undervisningsopplegg ved å bruke tre bøker som brukes ved innføringskurs i programmering. Dette med bakgrunn i at innholdet i bøker ofte blir førende for undervisningsopplegg (2.5 Bøker) og progresjonen som brukes (2.6 Forelesning). Jeg bruker stoffet som bakgrunn for å sammenligne (7.1 Andre undervisningsopplegg versus min oppgave) disse undervisningsopplegg med vårt eksperiment, slik det er beskrevet i 6 Begrunnelse for eksperimentet, 8 Gjennomføring av eksperimentet og 9 Drøfting og funn. Jeg tar først for meg hver bok, før jeg gir min vurdering av disse.

5.1 Java Gently

Boka Java Gently (Bishop, 1998) introduserer klassebegrepet ved en figur og tekst på en tredjedels side allerede på side 12 i introduksjonskapitlet.

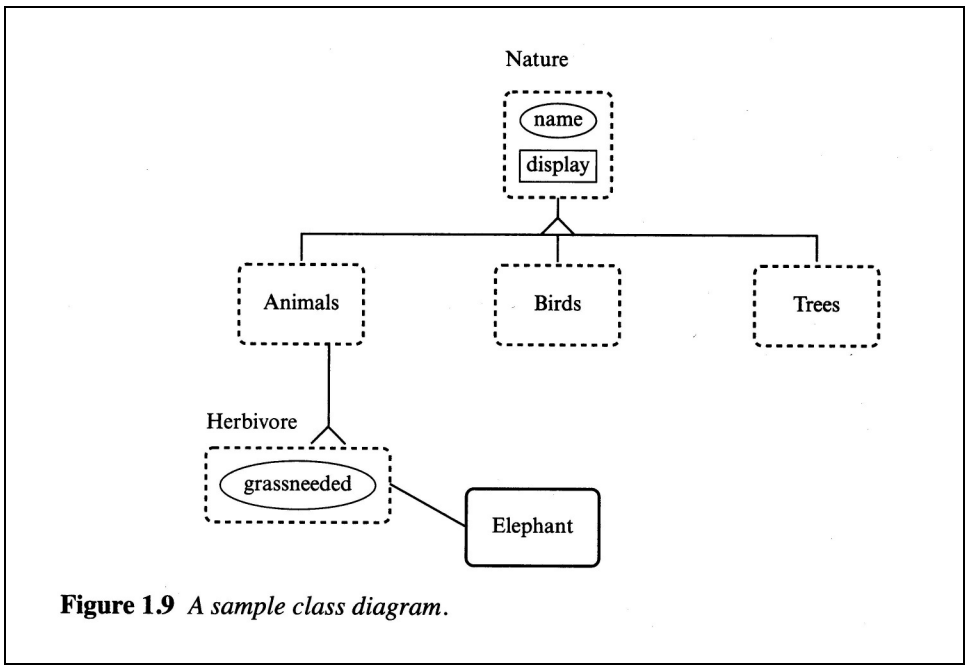


Figure 1.9 *A sample class diagram.*

Figur 6: Eksempel på klassediagram.

Figuren viser et enkelt klassediagram basert på OMT(object modelling technique) notasjon. Vi kan se at Nature inneholder et navn og en del som kan vise navnet. Vi kan allerede her merke oss at forfatteren introduserer et behov for å avlede nye klasser fra en hovedklasse med å bruke Animals, Birds og Trees som tre alternative klasser under Nature. Planteeter (Herbivore) er en klasse som blant annet inneholder mengden av gress som trenges for slike dyr. Forfatteren bruker notasjonen for å skille mellom hvilke klasser som kan finnes og hvilke objekter som virkelig

eksisterer i et system. Hun bruker stiplete bokser som viser klasser som kan eksistere, mens i dette systemet er Elephant det eneste objektet som finnes.

Denne korte innledningen har ingen eksempler eller tekst som det bygges videre på. Med bakgrunn i tekstens utforming og at det heller ikke er noen avklaring på forskjellen mellom begrepene klasse og objekt, oppfatter jeg teksten som en kort informasjon til en som er familiær med bruken av begrepene.

Opplegget i boken er at i kapitel 2 innføres en mer utdypende beskrivelse av begrepene klasse og objekt. Begrepene innføres ved å bruke modeller. Notasjonen hjelper studentene til å holde oversikt over hvilke begreper som illustres.

Kapitlet starter med to programeksempler: Et lite som skriver ut ”Welcome to Java” og et enkelt som lager et lite skjermbilde med de olympiske ringer. Resten av kapitlet bygger ikke videre på disse eksemplene. Innføringen av klasse og objektbegrepet skjer i kapitel “2.2 Fundamentalts of object-oriented programming”. Der står det: ”A Java **program** consist of a set of one or more inter-dependent classes as shown in Figure 2.2. **Classes** are a means for describing the properties and capabilities of objects in real life that the program has to deal with” (Bishop, 1998).

Videre i teksten kommer det ”Once we have a class, we can create many objects of that same class.” (Bishop, 1998).

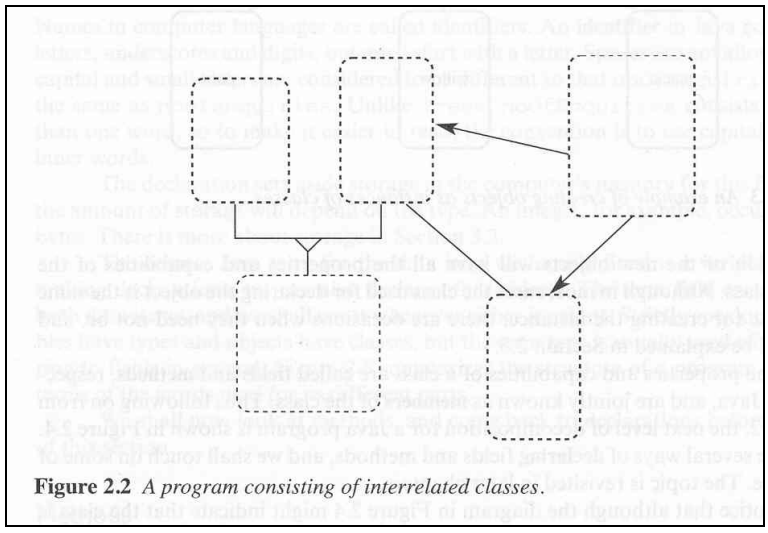
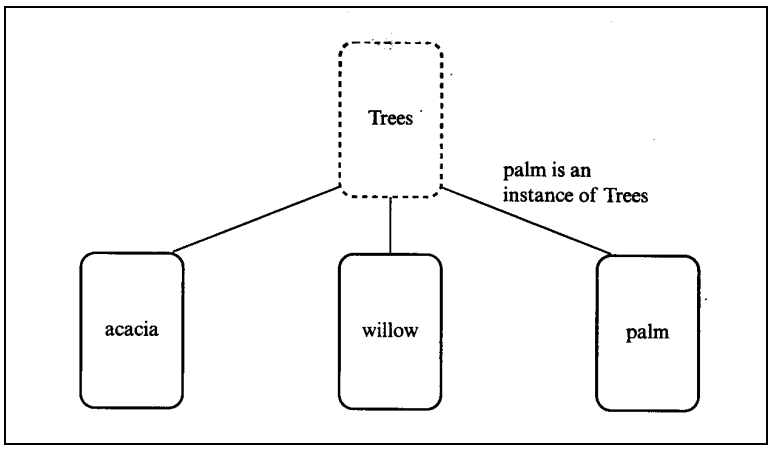


Figure 2.2 *A program consisting of interrelated classes.*

Figur 7: Relasjoner mellom klasser.

Figuren illustrerer teksten og viser på et overordnet nivå at det er assosiasjoner mellom klassene i et program.

”An **Object** is a concrete realization of a class description” (Bishop, 1998). Dette illustreres med en figur:



Figur 8: Arv.

Så følges dette opp med noe tekst om deklarasjoner og metoder. Først litt senere kommer programkode satt sammen til program. Teksten som omhandler kodebitene er laget generell og gir lite knytting til klasse og objektbegrepene. Det må dog sies at senere kommer et eget kapitel ”2.5 Using packages, classes and objects” og her er det en del programkode.

I kapittel 9 på side 267 kommer abstraksjon og arv. Her beskrives det å generere objekter ut i fra klasser, ved å først definere de tre ordene komposisjon, abstraksjon og arv.

”**Composition**, in other words, creating an object in a class based on another class.” (Bishop, 1998)

“**Abstraction** enables a class or method to concentrate on the essentials of what it is doing-its behaviour and interface to the world – and to rely on the details being filled in at a later stage.” (Bishop, 1998)

“With **inheritance**, we concentrate on defining a class that we know about, and leave open an option to define additional versions of it later. These versions will inherit the properties and characteristics of the original class, and therefore can be smaller in themselves, and neater. In this way we build up hierarchies of classes and can focus changes and additions at the right level.” (Bishop, 1998)

Disse definisjonene kobles så opp mot figuren:

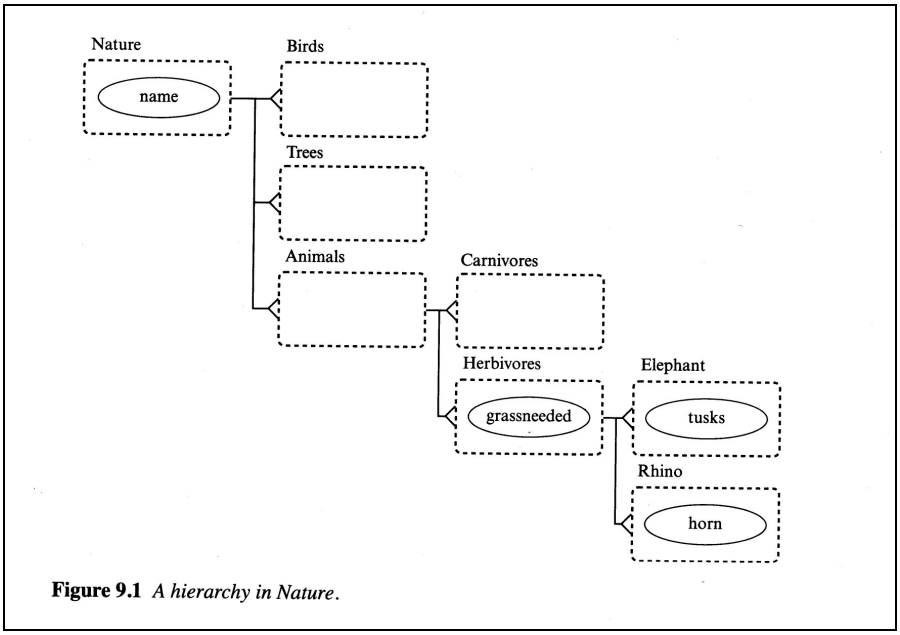
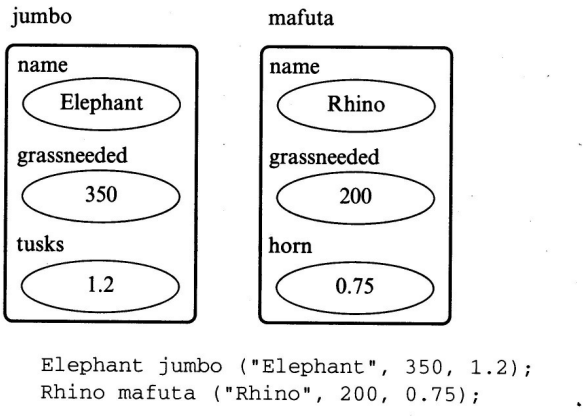


Figure 9.1 *A hierarchy in Nature.*

Figur 9: Hierarki i naturen.

Bakgrunnen for denne figuren blir beskrevet ved å referere til kapitel 1, se: (Figur 6: Eksempel på klassediagram.). Siden forfatteren under definisjonen av arv, har argumentert for å plassere felt og metoder så høyt i hierarkiet som mulig, brukes dette videre og knyttes mot figuren. ”Now the field holding the name of the animal could be rigth in the top class,” og data om lengden på støttenner plasseres i klassen Elephant fordi dette er spesifikt for elefanter. Mens gressmengden som trenges for en dag står under Herbivores.

Så brukes disse deklarasjonene til å generere to objekter.




```
Elephant jumbo ("Elephant", 350, 1.2);
Rhino mafuta ("Rhino", 200, 0.75);
```

Figure 9.2 *Object declarations in the Nature hierarchy.*

Figur 10: Objektdeklarasjon i et hierarki i naturen.

Boken påpeker i teksten hvilke felt som kommer fra klassene i bunnen og hvilke som kommer fra klasser lenger oppe i hierarkiet.

Det neste steget til forfatteren er på side 277 der begrepet arv knyttes mot programeksempler.

Som vi kan se har forfatteren brukt tre steg underveis. Et introduksjons steg, for så i steg to å prinsipielt beskrive klasse og objekt ved hjelp av modeller. Steg to benyttes også til å plassere felt og metoder i sammenheng med de to begrepene. I tredje steg knyttes kunnskapen mot programsetninger. Senere i boken brukes klassediagrammer som illustrasjoner knyttet opp mot programsetninger.

Selv om boken har disse introduksjonene opplever jeg at i resten av boken er det fokus på det å lære programmering og at den tar utgangspunkt i en prosedural tankegang (4.7 Programmerings-paradigmer).

5.2 JAVA som første programmeringsspråk

I boken: JAVA som første programmeringsspråk (Mughal, Hamre og Rasmussen, 2003), står det i forordet til boken: ”Hovedtema i boken er objektorientert programmering (OOP).… Boken er strukturert rundt objektorientering …Boken legger vekt på objektorienteringskonsepter og viser deretter hvordan disse implementeres…”

Allerede på side tre introduseres objektbasert programmering med en tekst på en halv side. Forfatterne tar utgangspunkt i:”…… handlingene som må utføres for å lage en omelett: Åpne kjøleskapet, ta ut en eggkartong, åpne eggkartongen, ta ut to egg, lukke kartongen, sette kartongen tilbake i kjøleskapet, lukke kjøleskapet, skru på stekeplaten, osv. I denne sammenhengen kan vi betrakte kjøleskapet, eggkartongen, eggene og stekeplaten som forskjellige objekter. Handlingene som kan utføres, er operasjoner tilknyttet disse objektene. Det er for eksempel mulig å *åpne* en eggkartong, men det er ikke mulig å *åpne* en stekepanne.

Objektbasert programmering (OBP) består av å beskrive større arbeidsoppgaver ved hjelp av operasjoner som utføres på objekter…. ha objekter både for helt konkrete gjenstander som bøker, tidsskrifter, lydkassetter o.l. og for mer abstrakte konsepter som selve utlånet og den informasjonen som behøves om hver låntager. I det siste tilfellet er låntageren selv et konkret objekt, dvs. en person, mens den informasjonen som biblioteksprogrammet trenger, er abstrakt.

Programmer har vanligvis flere objekter av samme type. La oss gå tilbake til kjøkkenet. Vi kan ha flere objekter som representerer kakestykker. Hvert kakestykke kan manipuleres uavhengig av de andre, men de har alle fellestrekk, slik som muligheten til å spise av det. Vi sier at vi har forskjellige *klasser* av objekter. Alle kjøleskapobjekter utgjør en klasse, mens alle kakestykkeobjekter utgjør en helt annen klasse.

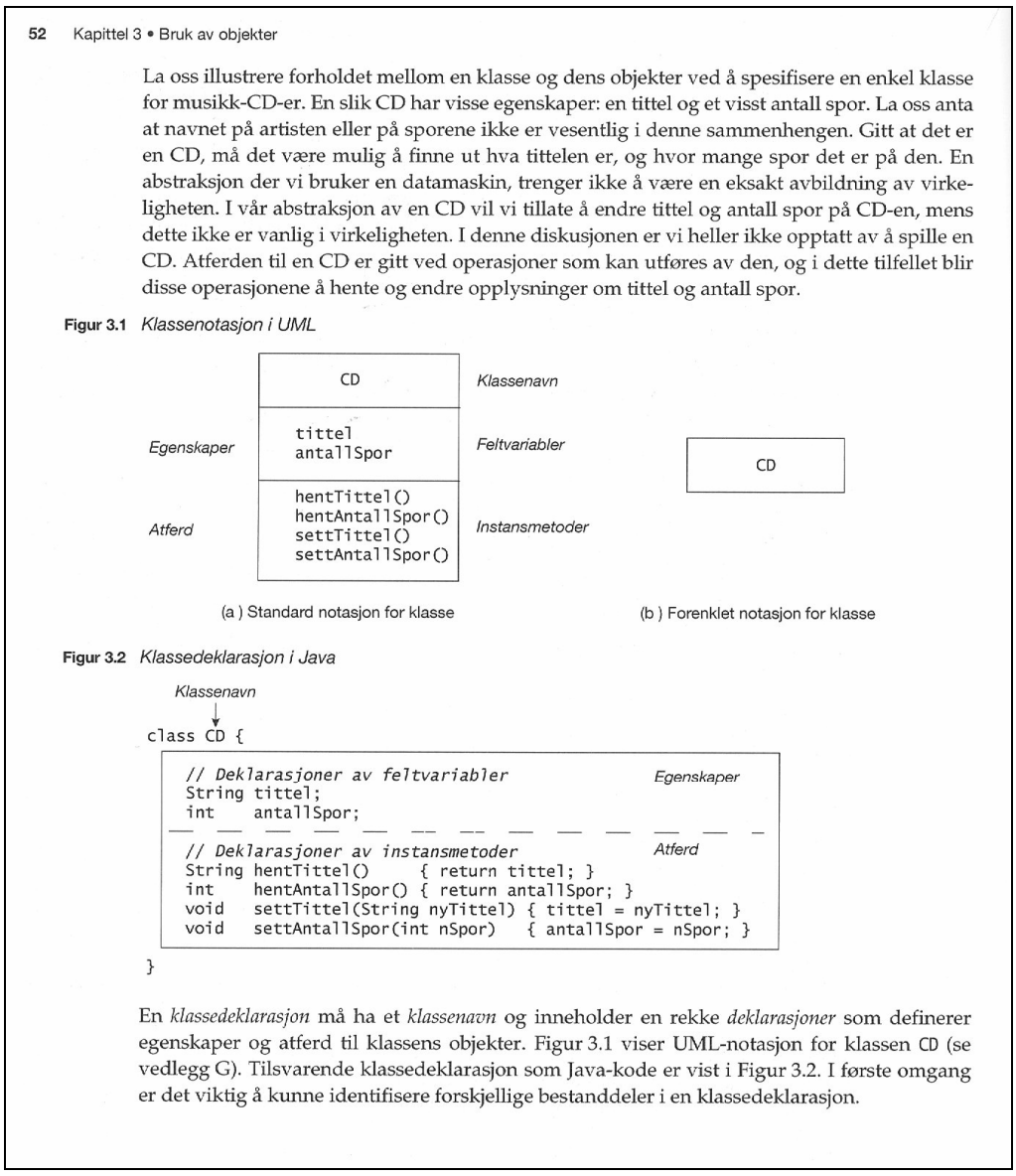
En klasse blir definert ved å beskrive hva som er særegent for objektene av denne klassen, og hvilke operasjoner som kan utføres på dem. Et program kan både benytte eksisterende klasser og definere egne klasser.

Dette er hele teksten og det brukes ingen figurer for å illustrere teksten. Innfallsvinkelen med å ta utgangspunkt i funksjonalitet for å beskrive begreper som objekt er lite i tråd med intensjonene slik de blir beskrevet i forordet til boken (se ovenfor). Deretter kommer et enkelt Java-program som skriver uten enkel tekst. Det er også med litt tekst som beskriver hvorledes programkoden skal skrives.

På side 51 ser forfatterne nærmere på forholdet mellom en klasse og dets objekter. ”Vi mennesker bruker abstraksjoner for å håndtere mangfoldet som møter oss i hverdagen. En *abstraksjon* betegner vesentlige egenskaper ved et objekt, slik at vi kan skille det fra andre typer objekter…. Vår evne til å lage en abstraksjon av hva vi oppfatter som kjøretøy, hjelper oss til å betrakte disse objektene som kjøretøy. Fargen eller modellen er ikke så vesentlig for å oppfatte at et objekt er et kjøretøy, men at det har hjul, motor, plass til sjåfør og kan kjøres, er avgjørende.

Vi skal representere abstraksjoner fra virkeligheten (for eksempel abstraksjoner for kjøretøy, kart, møter, datoer) i datamaskinen. Vi skal realisere abstraksjoner ved hjelp av klasser i Java. En *klasse* beskriver bestemte typer objekter. Den spesifiserer *egenskaper* og *atferd* til disse objektene”.

Deretter kommer det en tekst og figurer som viser klassenotasjon i UML (Unified Modeling Language) og klassenotasjon i Java. Jeg bruker denne siden også som eksempel på en side i boken:



52 Kapittel 3 • Bruk av objekter

La oss illustrere forholdet mellom en klasse og dens objekter ved å spesifisere en enkel klasse for musikk-CD-er. En slik CD har visse egenskaper: en tittel og et visst antall spor. La oss anta at navnet på artisten eller på sporene ikke er vesentlig i denne sammenhengen. Gitt at det er en CD, må det være mulig å finne ut hva tittelen er, og hvor mange spor det er på den. En abstraksjon der vi bruker en datamaskin, trenger ikke å være en eksakt avbildning av virkeligheten. I vår abstraksjon av en CD vil vi tillate å endre tittel og antall spor på CD-en, mens dette ikke er vanlig i virkeligheten. I denne diskusjonen er vi heller ikke opptatt av å spille en CD. Atferden til en CD er gitt ved operasjoner som kan utføres av den, og i dette tilfellet blir disse operasjonene å hente og endre opplysninger om tittel og antall spor.

Figur 3.1 *Klassenotasjon i UML*

Figur 3.2 *Klassedeklarasjon i Java*

```
Klassenavn
class CD {
  // Deklarasjoner av feltvariabler                Egenskaper
  String tittel;
  int    antallSpor;
  // Deklarasjoner av instansmetoder               Atferd
  String hentTittel()     { return tittel; }
  int    hentAntallSpor() { return antallSpor; }
  void   settTittel(String nyTittel) { tittel = nyTittel; }
  void   settAntallSpor(int nSpor)   { antallSpor = nSpor; }
}
```

En *klassedeklarasjon* må ha et *klassenavn* og inneholder en rekke *deklarasjoner* som definerer egenskaper og atferd til klassens objekter. Figur 3.1 viser UML-notasjon for klassen CD (se vedlegg G). Tilsvarende klassedeklarasjon som Java-kode er vist i Figur 3.2. I første omgang er det viktig å kunne identifisere forskjellige bestanddeler i en klassedeklarasjon.

Figur 11: Eksempel på en side.

Som en ser av figuren brukes UML får å illustrere klassen, og denne modellen stemmer overens med koden som står på samme siden.

Jeg skal nå se litt nærmere på teksten.

Klassebegrepet innføres på side tre med en klar tekst som sier at ”alle kjøleskapobjekter utgjør en klasse, mens alle kakestykkeobjekter utgjør en helt annen klasse”. Dette er klart og bør være rimelig greit å få en forståelse av for en nybegynner i programmering. Også fortsettelsen er lett å forstå ” En klasse blir definert ved å beskrive hva som er særegent for objektene av denne klassen, og hvilke operasjoner som kan utføres på dem.” Sammenligner vi denne teksten med det som kommer i kapittel tre, ser jeg at dersom vi bare holder oss til klasse og objektbeskrivelsene, stemmer dette godt over ens.

Det som gjør at deler av teksten i kapittel tre er uklar og kan bidra til misoppfatninger, er at forfatterne innfører ordet ”*abstraksjon*”. Jeg skal prøve å vise dette litt nærmere nedenfor.

Siden et hovedtema i boken er objektorientert programmering, bruker forfatterne ordet abstrakt om noe som er vesentlig ved et objekt. Den første definisjonen er på side 3: ”I det siste tilfellet er låntageren selv et konkret objekt, dvs. en person, mens den informasjonen som bibliotekprogrammet trenger, er abstrakt.” Her kommer det ingen eksempler som kan hjelpe til å bygge en forståelse hos nye studenter. Derfor er det naturlig at en nybegynner oppfatter at konkrete objekter er objekter som en kan ta på, mens abstrakte objekter er den ”informasjonen” som bibliotekprogrammet trenger. Forklaringen sier jo at alle objektrepresentasjonene i datamaskinen er abstrakte. Denne forklaringen er lite i overensstemmelse med begrepsbruken som brukes av andre i fagmiljøet.

I ”3.1 Innføring i objektmodellen” heter det ”Målet med dette avsnittet er å innføre noen viktige begreper uten å gå i detaljer.” Dette målet prøver forfatterne å følge opp, forså vidt i hele boken, med å ha mange definisjoner. Noen av disse blir lite heldige beskrivelser og et eksempel er ordet ”*abstraksjon*”. Beskrivelsen binder ordet til ”vesentlige egenskaper” ved et objekt. Vesentlige egenskaper beskrives så som noe som brukes for å skille en type objekter fra en annen type. Dette er ved nøye lesning korrekt, men vil lett kunne misforstås av en student som har brukt litt databaser. Jeg har opplevd at jeg og tre andre som jeg har latt lese det aktuelle avsnittet oppfattet at abstraksjon i teksten nærmest beskrev noe som i databaser kalles en identifikator. Dette kan indikere at definisjonene lett kan misforsåes.

For boken generelt er de enkelte definisjoner skrevet i et språk som krever en innsikt i måter å uttrykke akademiske definisjoner på, og derfor er teksten langt fra det språk som vi kan forvente at målgruppen er vant med. Jeg vil hevde at språket er mer egnet for personer som er drevne i objektorientert programmering. Dette med bakgrunn i den nokså akademiske form som for enkelte vil være grei, mens store deler av målgruppen vil være avhengig av støtte som forelesninger og veiledning kan gi. Her tenker jeg spesielt på det å kunne assosiere til en for studentene kjent verden. Med andre ord opplever jeg at boken er mer egnet for studenter med programmerings og studieerfaring enn for nybegynnere.

Nye studenter vil tradisjonelt være vant til at et objekt er en fysisk gjenstand (blyant, hus, bil osv) og at objektet ikke kan utføre noe, det er en død ting. Når forfatterne begynner å snakke om objekter som om de har hendelser, oppstår det en konflikt i forståelsen boken legger til grunn og den erfaring som nybegynnere i programmering kan forventes å ha. De som jobber med undervisning tar ofte for lett på denne konflikten i oppfatning.

Boken tar utgangspunkt i hendelser for å finne hvilke objekter som skal finnes i systemet det er lite i overensstemmelse med forskningsresultater som at overgangen fra prosedural programmering (4.7 Programmerings-paradigmer) til objektorientert programmering er en vanskelig og tidkrevende prosess (Bergin, (2000), Stroustrup, (1994), Lattanzi og Henry, (1996)).

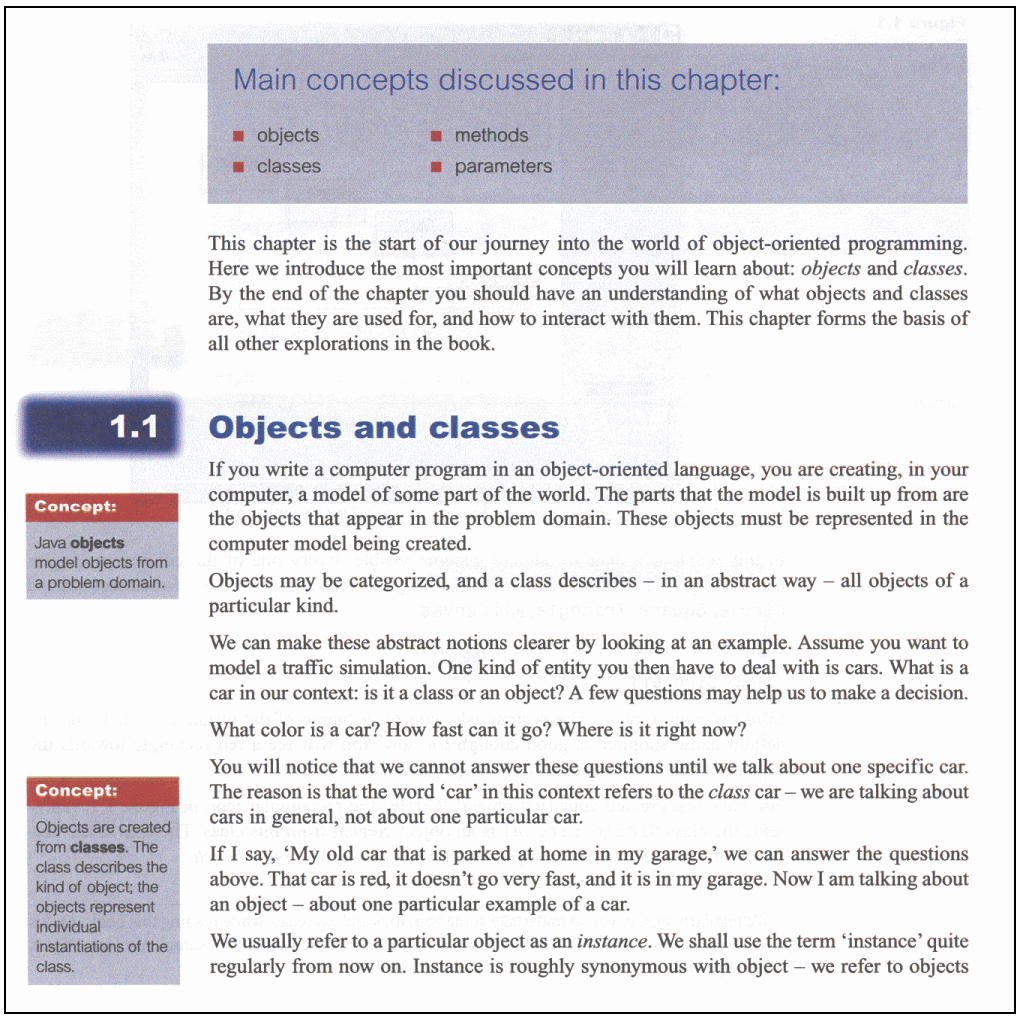
Når erfarne prosedural programmerere er i lærefasen for å lære seg objektorientert programmering, vil de naturlig prøve å løse problemer ved å dekomponere funksjoner og ikke ved å oppdage objekter.(Bergin, 2000). Begrunnelsen ovenfor kan være et godt argument for at nybegynnere skal starte rett på å oppdage objekter, og så prøve å knytte handlinger til disse objektene. Dette blir ytterligere forsterket av at J. Bergin (2000) argumenterer for: At det er lettere å gå fra OO paradigmet til prosedural enn den andre veien. Dette fordi en lærer om funksjoner i OOP. Gode prosedurale program kan bli laget av OOP programmerere, fordi de helt naturlig vil partisjonere sine program inn i objektlignende enheter av funksjonalitet selv om de ikke har samme vertøyet som de er vant med fra OOP verden.

En måte å klargjøre begrepet objekt på kan være å bruke tre oppfattelser av ordet. En er den fysiske gjenstanden. To er vår mentale oppfattelse av objektet, og det tredje er den representasjonen som finnes i en datamaskin. Ved å klart skille disse tre oppfattelsene på et tidlig stadium for studentene, kan det lette deres jobbing med oppgaver.

Det er en styrke at boken prøver å bruke definisjoner av begreper i et akademisk formulert språk, men dette kan være vanskelig tilgjengelig for målgruppen. Enkelte av definisjonene er ikke i overensstemmelse med de andre bøkene. Jeg hevder at enkelte eksempler kan oppfattes lite konsistente av målgruppen.

5.3 Java BlueJ

Boken Objects First With JAVA, a practical introduction using BLUEJ, Second edition (Barnes og Kölling, 2005) er en bok som baserer seg på at studenten skal anvende verktøyet BlueJ som har et læringsmiljø som inneholder skjermbilder og metoder som eksemplifiserer begreper som brukes i Java programmering. Allerede i bokas første kapittel introduseres begrepene objekt og klasse.



Main concepts discussed in this chapter:

- objects
- classes
- methods
- parameters

This chapter is the start of our journey into the world of object-oriented programming. Here we introduce the most important concepts you will learn about: *objects* and *classes*. By the end of the chapter you should have an understanding of what objects and classes are, what they are used for, and how to interact with them. This chapter forms the basis of all other explorations in the book.

1.1 Objects and classes

Concept:

Java **objects** model objects from a problem domain.

If you write a computer program in an object-oriented language, you are creating, in your computer, a model of some part of the world. The parts that the model is built up from are the objects that appear in the problem domain. These objects must be represented in the computer model being created.

Objects may be categorized, and a class describes – in an abstract way – all objects of a particular kind.

We can make these abstract notions clearer by looking at an example. Assume you want to model a traffic simulation. One kind of entity you then have to deal with is cars. What is a car in our context: is it a class or an object? A few questions may help us to make a decision.

What color is a car? How fast can it go? Where is it right now?

Concept:

Objects are created from **classes**. The class describes the kind of object; the objects represent individual instantiations of the class.

You will notice that we cannot answer these questions until we talk about one specific car. The reason is that the word 'car' in this context refers to the *class* car – we are talking about cars in general, not about one particular car.

If I say, 'My old car that is parked at home in my garage,' we can answer the questions above. That car is red, it doesn't go very fast, and it is in my garage. Now I am talking about an object – about one particular example of a car.

We usually refer to a particular object as an *instance*. We shall use the term 'instance' quite regularly from now on. Instance is roughly synonymous with object – we refer to objects

Figur 12: Første side.

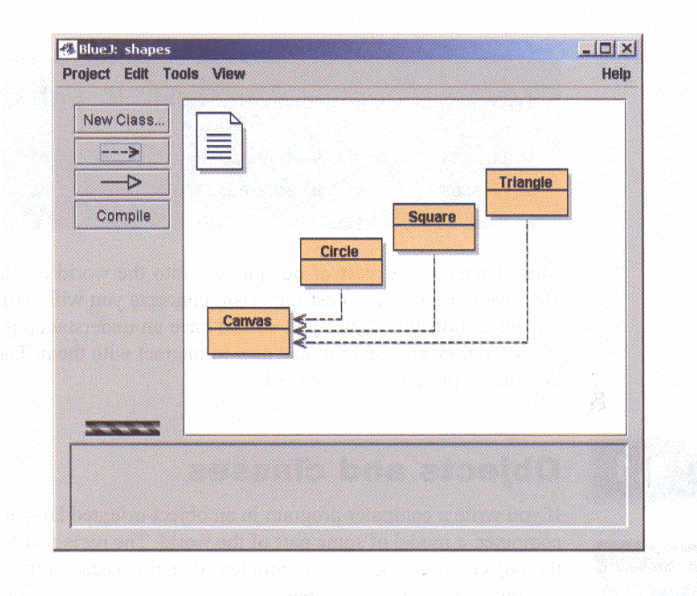
Vi ser av teksten at forfatterne ønsker å gi leseren en forståelse av objekter og klasser samt hvorledes vi kan ha en interaksjon med disse. Når det gjelder læring av objektorientert programmering er strategien i boka at byggingen av kunnskap i programmering tar utgangspunkt i studentens tidligere erfaring fra situasjoner i dagliglivet. Boka er altså basert på en konstruktiviskisk tankegang (6.6 Konstruktivistisk). Med dette som basis skal vedkommende erfare gjennom praktiske øvelser hvorledes begrepene kommer til anvendelse i programmeringen. Her brukes i stor utstrekning modellfigurer som illustrasjoner for at studenten ikke bare skal ha en tekstlig beskrivelse å forholde seg til, men ha visuelle bilder som teksten knyttes til. Etter at studenten er blitt kjent med begrepene behandles de mer utførlig med å knytte mer kode til den erfaring studenten allerede har. Da kommer også mer eksakte definisjoner. Jeg beskriver nedenfor progresjonen i boka for å lære objektorientert programmering.

Allerede i første avsnitt knyttes begrepet objekt opp mot problemområdet med "Concept: Java **objects** model objects from a problem domain." (Barnes og Kölling, 2005) Begrepet objekt knyttes således mot leserens oppfattelse av verden. På en like enkel måte beskrives klassebegrepet med " Concept: Objects are created from **classes**. The class describes the kind of object; the objects represent individual

instantiations of the class." (Barnes og Kölling, 2005). For at leseren skal få en mer direkte erfaring med hvorledes begrepene brukes, har forfatterne et eksempel med trafikksimulering. De stiller spørsmålet om hva en bil er i vår sammenheng: Er det et objekt eller en klasse? Med noen enkle spørsmål om farge, maksimal hastighet, hvor er bilen akkurat nå?, får leseren en klar oppfattelse av at for å svare på disse spørsmålene må vi snakke om en spesifikk bil, det vil si et objekt.

Isteden for å komme med en akademisk utredning om begrepene objekt og klasse legger de opp til at leseren skal bygge kunnskap ved å erfare begrepene ved hjelp av verktøyet BlueJ.

BlueJ har en verktøykasse som inneholder en del skjermbilder som gir muligheter for å prøve å generere en instans av en klasse dvs et objekt.



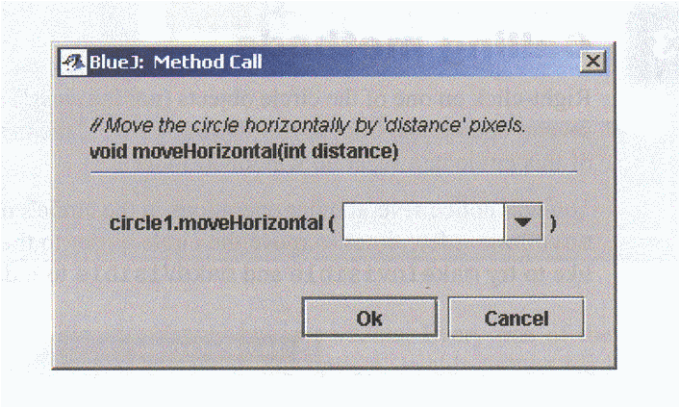
Figur 13: Skjermbilde for å generere en instans av en klasse.

Figuren har fire klasser, ved å klikke på for eksempel Circle får vi muligheten til å generere en instans av klassen Circle. Nå vil brukeren erfare at det spørres etter et navn på instansen, når dette er innfylt trykkes OK og objektet dukker opp som et rektangel nederst i bildet.

Etter dette introduserer boken bruk av metoder ved at brukeren skal høyreklikke på et objekt og får se noen ferdig skrevne metoder som for eksempel makeVisible, moveRigth og moveDown. Ved å klikke på metodene kan brukeren se en visualisering av resultatet av at metoden utføres. Dette kan sees i et eget vindu hvor objektet visualiseres.

Figur 14: Eksempel på visualiseringsvindu.

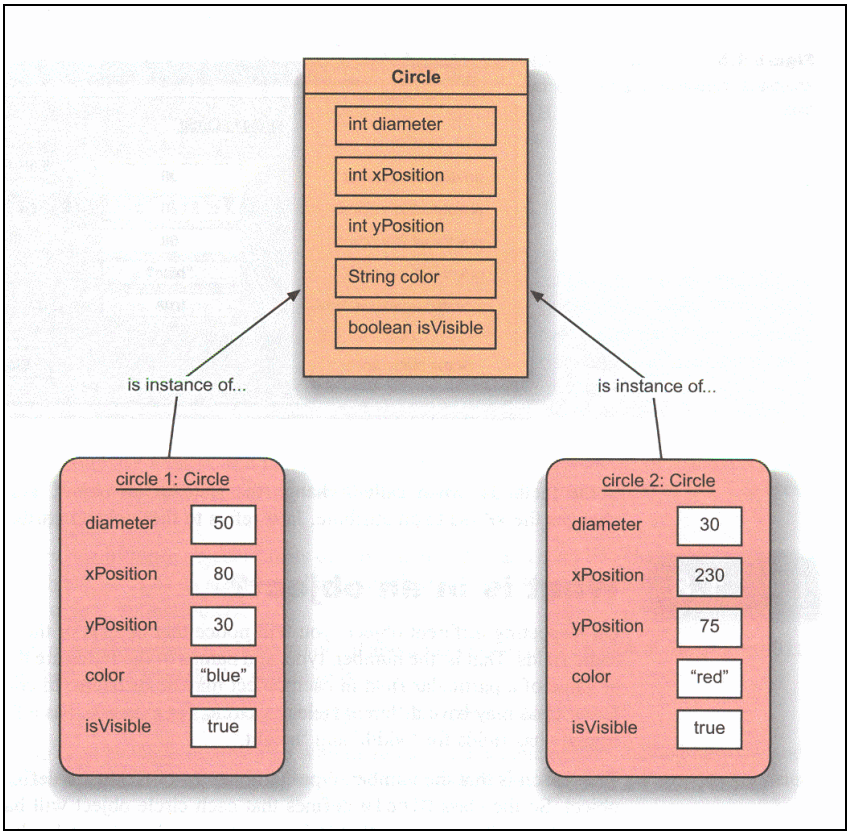
Samme teknikk brukes når begrepet parameter innføres. Dette gjøres ved at du klikker på et metodekall som krever parameter og det kommer da et vindu som ber om parameter.



Figur 15: Eksempel på metodekall med parameter.

Når verdien er fylt ut og du har trykket på OK knappen, kan du se resultatet i visualiseringsskjermbildet.

Allerede på side 9 kommer en mer utfyllende beskrivelse av begrepet objekt. Her beskrives klasse som en mal som brukes for å generere objekter.



Figur 16: Klasse og objekt i samme figur.

Denne figuren er et eksempel på at bruk av to begreper som klasse og objekter kan brukes for å beskrive relasjonen mellom begrepene. Det å blande begrepene slik er ikke etter standard UML-notasjon. Styrken i figuren er at malen beskriver hvilke parameter som objekter som genereres fra klassen skal ha, samtidig som de illustrerer eksempler på to objekter med verdier parametrene kan få.

Så fortsetter forfatterne med å beskrive hvorledes brukeren kan ha en interaksjon med objekter og at objekter kan kommunisere med andre objekter. Dette gjøres med å anvende øvelser som bruker de ferdige metodekall og parameter.

Allerede på side 13 får studenten øvelse i å bruke objekter som parameter til metoder i andre objekter. Så hele kapittel 1 brukes til at studenten skal få en klarere formening om hvorledes begrepene objekt og klasse brukes i programutvikling. Dette gjøres ved en minimalistisk teoretisk innføring i begrepene.

Først i kapittel 2 ”Understanding class definitions” behandles kodebitene som trenges for å definere en klasse.

Jeg finner at boken fokuserer på å bygge kunnskap på først å erfare og så knytte teoretisk kunnskap til disse erfaringene. Boken legger opp til at brukeren først skal få en viss erfaring i begrepene klasse og objekter og hvorledes objekter kan instrueres til å utføre visse handlinger. I denne innføringsfasen beskriver teksten bare nok til at brukeren har en viss forståelse av begrepene, og slik at studenten kan jobbe med øvelsene som ligger i BlueJ. Arbeidet med øvelsene kan gjøres uten at

brukeren behøver å ha noen kunnskap i programkode, kompilering og kjøring av programmer slik tradisjonelle programmeringskurs krever. Den vesentlige årsaken til dette er at læringsmiljøet i BlueJ er vesentlig enklere en det som finnes i tradisjonell programmering. Brukeren kan i øvelsene ha en interaktiv kommunikasjon med objektene, det vil her si påvirke og umiddelbart se resultatet. Et eksempel på dette er visualisering av klassestrukturen med hjelp av en UML-lignende modell (Figur 13: Skjermbilde for å generere en instans av en klasse).

Først lenger ut i boken knyttes det nødvendig teori til opparbeidet kunnskap. Et eksempel på dette er at kildekode først kommer i kapittel 2 når studenten har anvendt klasser og objekter en god del.

Boken legger opp til et vokabular og språkform som jeg mener er godt tilpasset målgruppen. Den er også en god støtte til å skaffe seg kunnskap (6.6 Konstruktivistisk) om objekter og klasser uten at det kreves erfaring i akademiske måter å formulere språket på.

5.4 Drøfting av bøker

Disse tre bøkene har tre helt forskjellige pedagogiske opplegg. Jeg skal her prøve å beskrive min vurdering av disse tre bøkene. Det at bøkene er såpass forskjellig i opplegg kan ha utspring i at forfatterne oppholdt seg i tre forskjellige kontinenter (Sør-Afrika, Norge og Australia).

Boken Java Gently baserer seg på å bygge på at forfatteren bruker en begrepsverden som høyst sannsynlig er kjent for studenten (6.6 Konstruktivistisk). Disse begrepene er fra en verden utenfor programmering, og så knyttes disse mot de objekt-orienterte begrepene som forfatteren ønsker å innføre. Dette gjøres ved å legge opp til å bygge en overordnet forståelse for hvorledes begrepene og deres sammenheng anvendes i programmering. Som eksempel bruker jeg: Objekt og klassebegrepet innføres ved at forfatteren prøver å bygge på kunnskapen studenten har fra før, ved å bruke eksempler fra dyreriket i Afrika. Det kan være at det å bygge på denne kunnskapen fungerer godt for personer som føler at de kjenner eksemplene fra før, hvor godt dette passer i en storby i en annen verdensdel som for eksempel Sør-Amerika, eller fra en kultur hvor studentene tradisjonelt er vant til å bruke en annen oppbygging av dyreriket er et spørsmål. Jeg oppfatter det slik at for en norsk student vil måten objekt og klassebegrepene introduseres på kunne fungere etter det som jeg oppfatter er hensikten til forfatteren. En av grunnene til dette er at figurene som brukes som illustrasjoner er enkle og oversiktlige.

Selv om boken starter med en fyldig og klar beskrivelse av objekt-orienterte begreper, opplever jeg ikke at boken fokuserer på objekt-orientering, men derimot på det å lære bort grunnleggende tenking i det å programmere en datamaskin, og da med fokus på Java. Objektbegrepet opplever jeg som mer perifert og da med bakgrunn i at etter den innledende innføringen så er det prosedurale strategier (4.7 Programmerings-paradigmer) som dominerer i teksten og i eksemplene.

I boken Java som første programmeringsspråk sier allerede tittelen at også denne boken er skrevet for personer som skal lære å programmere. I omtalen bak på boka

står det at det er objektorientering som er hovedtema. Dette er ikke min opplevelse av boken. Jeg opplever at boken fokuserer på nøyaktige definisjoner av begrepene og detaljer i Java programmering, mer enn forståelse for sammenhengen mellom begrepene og hvorledes de skal brukes i programmering. Tilnærmingen til stoffet er å gi formelt riktige definisjoner og program kode som skal beskrive hvorledes de brukes i programmering. Med dette mener jeg her at for en nybegynner vil boken oppleves som om at du lærer elementer for etterhvert å bygge en kunnskap om sammenhengen mellom begrepene.

I boken Java BlueJ starter forfatterne med å introdusere begrepene objekt og klasse og binde disse begrepene opp mot den erfaring som studenten har fra tidligere. Boken legger opp til at studenten skal vinne erfaringer med begrepene ved at det anvendes et verktøy som har mange pre-programmerte muligheter. Disse mulighetene gjør at studenten meget enkelt kan prøve øvelser, og å se virkningen på skjermen. Senere i boken kommer mer utfyllende kode og teori.

Java Gently og Java BlueJ bruker begge som basis at studenten har noe kunnskap om den verden som er utenfor programmering. Denne kunnskapen brukes i tekst og modeller for å illustrere sammenhengen i begrepene. Dette gjøres blant annet ved at det brukes modeller som inneholder både klasser og objekter i samme modell. En forskjell er at jeg opplever at Java Gently fokuserer mer på teksten, mens Java BlueJ fokuserer mer på illustrasjoner og bruker teksten som supplement. Boken ”Java som første programmeringsspråk” bruker en helt annen innfallsvinkel. Den tar som beskrevet ovenfor, utgangspunkt i operasjoner/handlinger og presenterer så programkodeelementer. At handlinger er sentrale fremkommer også av at det brukes sekvens og hendelsesdiagrammer. Her finnes ingen klasse eller objektdiagram som kan illustrerer disse delene av teksten.

Arbeid med øvelser er i Java Gently og Java som første programmeringsspråk, basert på en tradisjonell arbeidsform som er vanlig i bransjen. Det vil si bruk av editor, kompilering og kjøring av programmer. Dette gjør at terskelen for å komme i gang blir høyere enn i Java BlueJ som har et mer utbygd verktøy, som spesielt i introduksjonsfasen gjør at studentene lett kan se på illustrasjoner av objekter og kan instruere disse. Når noe er lett å bruke kan det ofte være en hake med det, og her er det at brukeren er nokså avhengig av å bruke øvingsopplegget som boken legger opp til. Den norske boken har til dels samme problemet med å tilpasse øvingsopplegget. Her er grunnen at det er så detaljerte beskrivelser som er nært knyttet til eksemplene i boken, at det blir vanskeligere å tilpasse andre øvingsopplegg en det som boken legger opp til. Den boken som gir den største fleksibiliteten i valg av øvelsesopplegg er Java Gently. Teksten og illustrasjonene er på litt mer overordnet nivå, og gir dermed større frihet til å velge øvingsopplegg.

Dersom begreper som ”hands on” og ”learning by doing ” skal legges til grunn for et undervisningsopplegg, vil helt klart Java BlueJ være godt egnet, i det hele konseptet i boken legger opp til dette. Et vesentlig bidrag her er at det ikke fokuseres på detaljer i starten, men at studenten kommer direkte i gang med øvelser som skal illustrere begrepene klasse og objekt. De andre to bøkene er mer tradisjonelle, i det de presenterer stoffet for så å legge opp til at det skal jobbes med øvelser.

Hvor egnet bøkene er som oppslagsverk er avhengig av hva slags kunnskap som er målet. Dersom det er oversikt over begrepene og innsikt i hvorledes de kan modelleres er det Java Gently og Java BlueJ som på grunn av sin tilnærming vil kunne fungere. Dersom det er en overordnet kunnskap om objekter og hvorledes disse kan opprettes og instrueres, er Java BlueJ klart tydeligst i sin presentasjon. Men dersom leseren er erfaren programmerer og syntaks i programmet er målet, vil den norske boken fungere på grunn av sin detaljrikdom.

Kölling (2005) har gjort en undersøkelse av 41 bøker som introduserer programmering. Arbeidet hans kategoriserte de første programmerings ekseplene i bøkene. Han har funnet at inntil år 2000 brukte de aller fleste bøker *Hello, word* eksemplet eller tilsvarende korte statiske tekster. Etter dette og til og med 2002, hevdet nesten alle bøker at de fulgte en innfallsvinkel som i Norge omtales som objektorientering først. Men som Köllings undersøkelse viser: Som regel bidro de ikke til at kunnskap om objektorientering ble overført til studenter. Syntaksen som ble brukt brukte objekter, men eksemplene var slik konstruert at ingen hadde mulighet til å lære noe om objekt fra de første eksemplene (det vil si: Bare syntaks).

6 Begrunnelse for eksperimentet

Jeg skal prøve å relatere vårt opplegg til en del begreper som er presentert tidligere, samt trekke inn en del sentrale begrep innen læringsteori.

6.1 Mestring

Begrepet mestring er tidligere beskrevet nærmere i kapittel 4.2 Mestre, og blir der basert på Bandura (1986 og 1998), Pajares (2002) og Glanz et al. (2002) sine arbeider.

Jeg forutsetter at studenten fokuserer på det som vedkommende oppfatter skal være resultatet av arbeidet med en oppgave. Dersom studentens oppfattelse er at resultatet skal vær et program som virker, så er det naturlig at studenten har fokus på det som enklest kan bidra til å oppnå dette resultatet. Siden feilmeldinger ofte knyttes til syntaks og semantikk i programsetninger, vil det være naturlig at studenten velger å fokusere på årsaken til at feilmeldingene ble generert. Noe som også underbygger studentens oppfattelse, kan være at studenten ofte kan evaluere jobben som er gjort ved at det presenteres et svar eller reaksjon som kan observeres visuelt. For eksempel et tall eller noe som beveger seg på skjermen. Dette er det jeg vil beskrive som å være orientert mot resultatet.

”Hva om en snur kravet fra resultat til mestring/prestasjon?” (Finn Aamot, Olympiatoppen). Begrepet resultat henviser her til resultatliste i idrettssammenheng. Mestring omtaler jeg på to måter i forbindelse med å lære å programmere. Den ene er som tidligere beskrevet å mestre syntaks og semantikk i programsetninger. Den andre er det å mestre objektorienterte begreper, for der igjennom å kunne anvende kunnskapen i andre problemstillinger. I det siste ligger det også at kunnskap om begreper brukes som grunnlag for å overføre læring til nye oppgaver og begreper, samt generalisere.

Dette innebærer at aktiviteter må legges opp slik at fokus til studenten endres fra syntaks og semantikk i programsetninger, til at vedkommende får en forståelse av at, begrepene og deres sammenheng er et viktig moment for å kunne løse det aktuelle problem. Studenten må også kunne oppleve at det finnes et motiv for aktiviteten som oppfattes som nyttig. Dette siste øker behovet for at øvelsene skal tilfredsstille den enkelte students behov for utfordring og opplevelse av å lære noe nyttig.

I idretten snakkes det mye om behov for utfordringer og mestring. I dette ligger det at utøverne tøyer, det de opplever som sine grenser for hva de mestrer. For at en utøver skal oppleve å mestre noe, må utøveren ha en forståelse for hva som utføres og i hvilken sammenheng egenskapen skal brukes. Det at en utøver opplever at vedkommende mestrer noe, regnes som en vesentlig motivasjonsfaktor både for unger som driver med fysisk aktivitet og for utøvere som innefor sin idrett ligger

helt i verdenstoppen. Jeg bruker to eksempler for å vise hvorledes en i idretten jobber med dette.

Et eksempel på dette er alpinister som i treningen har lagt inn balansering på bom 2 meter over bakken. For at de skal ta utfordringen, må de oppleve at dette er en øvelse som de kan ha nytte av i utøvelsen i sin idrett. Nytteølelsen er motivasjonsfaktoren som skal drive de til å jobbe med utfordringen. Når de etter hvert klarer å balansere på bommen får det en følelse av å mestre en utfordring. Da legges det til ytterligere utfordringer i øvelsen, som hinder, ta i mot og kaste ball.

Et annet eksempel er idrettskoler som har fått mye omtale i de senere år. Olympiatoppen regnes som en ressurs når det gjelder å utvikle glede og interesse for å drive med en idrett, både for bredde og toppidrett, er helt klar på at dersom du skal utvikle interesse for en idrett må du ha en følelse av konkrete ting du mesterer. Finn Aamot fra Olympiatoppen holdt i februar 2004 foredrag for trenere og idrettsledere i Østfold. Han beskrev et eksempel hvor unger som hadde stor lyst til å bli like god fotballspillere som Ole-Gunnar Solskjær gikk på idrettsskole. Der måtte de starte med to – tre uker håndball, så noen uker med skigåing, så turn for etter hvert å komme til to - tre uker med fotball. Da hadde de mistet motivasjonen for idrett, fordi de ikke opplevde noe som de følte var nyttig i forhold til sin interesse. Denne formen å drive idrettskoler på er ganske utbredt i Norge i dag. En begrunnelse som brukes for dette opplegget er at ferdigheter fra en idrett kan overføres til andre idretter og gjøre at utøveren blir flinkere. Dette stemmer godt med at idretten i dag har solid kunnskap om at ferdigheter innefor en idrett kan overføres til helt andre idretter. Kunnskapen om overføringsverdier brukes bevisst av mange trenere i sitt arbeid både innenfor breddeidrett og helt opp til toppidrettsutøvere.

Olympiatoppen er negativ til den måten å drive idrettsskoler på, fordi ungene som ønsker å bli en ny Ole-Gunnar Solskjær ikke får tilstrekkelig dekket sitt behov for følelse av mestring av noe som de oppfatter som nyttig i relatert til sine ønsker og mål. Dette medfører at de går lei og slutter med idrettsaktiviteter.

Dersom disse erfaringene overføres til vårt problemområde skulle det indikere at aktivitetene må innholde problemstillinger som studenten kan gjenkjenne og oppleve som relevant for det faget vedkommende jobber med. Det medfører også at stoffet må legges opp slik at studenten etter en aktivitet, opplever å mestre det som var læringsmålet i aktiviteten.

Som jeg har omtalt tidligere er det et spørsmål om selvoppfattelsen er årsak eller effekt av akademiske prestasjoner. Uansett perspektiv på argumentasjonen ser vi at det henger sammen og derfor er det all grunn til å ha didaktiske opplegg som motiverer den ”gode følelsen”. Neste spørsmål blir da, skulle vi fokusere undervisningsinnsatsen, undervisningsmetode og akademiske strategier mot det å øke kompetansen hos studentene, eller skal vi fokusere på studentenes selvoppfattelse? Vi to som skulle holde kurset ble tidlig enig om at vi som utgangspunkt for planleggingen, ville basere oss på å bruke mestring av selvstendige oppgaver i en setting som studentene lett kunne kjenne igjen. Dette siden vi ønsket å gi en god følelse for å motivere til videre læring av programmering, samt at mitt angeliggende var å prøve veiledningen i en situasjon som studentene forhåpentligvis opplevde som positiv.

Faren med å bruke et slikt bevisst fokus på at mestring skulle gi en positiv opplevelse, er at det kan bli for mye fokus på enkelte studenter i gruppen og at andre føler at de ikke får tilstrekkelig hjelp. Dette er en svært vanskelig balansegang, fordi hjelpen ikke skal gjøre så mye at mestringfølelsen blir mindre enn den ellers ville ha vært. Forståelsen av de objektorienterte begrepene vi jobbet med, skal heller ikke bli forringet av at veileder forlater rollen som coach og går mer over i programmererrollen.

Vi stod ovenfor å velge mellom to måter å veilede på. Den første var å veilede på hva deltagerne skulle skrive av kode, med dette mener jeg at vi kunne fokusere på syntaksen og gi tips om hva de kunne skrive, for å få det som tidligere er beskrevet som suksessfølelse. Spesielt i de innledende fasene av det å lære programmering av datamaskiner kan det bli fokus på syntaks. Dette kan også sees i min vurdering av de tre lærebøkene. Vi hadde den oppfatningen at slik veiledning ville gi suksess på kort sikt, i det studentene løste den aktuelle oppgaven, men vi var skeptisk til om dette ville styrke begrepsforståelsen. Det vil si at neste gang de fikk et tilsvarende problem i en annen situasjon, ville de kanskje ikke ha en opplevelse av at de hadde erfaring som de kunne anvende for å komme et skritt nærmere en løsning. En annen måte å veilede på, kunne være at vi snakket om begrepene og prøvde å hjelpe studentene til de fikk en begrepsforståelse som gjorde at de lettere kunne kjenne igjen tilsvarende situasjoner. Dette gjorde vi ved å fokusere OO begrep og knytte disse opp mot en verden som studentene kjente, samt bygge erfaring på anvendelsen av begrepene hos studentene ved at de observerte konsekvensen av bruken. Vi valgte den siste måten.

6.2 Mesterlæring og prosjektarbeid

Begrepet mesterlære har jeg omtalt tidligere i kapittel 4.1 mesterlære.

Mesterlæring har visse likhetstrekk med tradisjonell programmeringsundervisning, ved at læreren underviser og viser eksempler. Dette siste kan for eksempel være å skrive programkode, kompilere eller kjøre programmet og samtidig vise via en projektor arbeidet som utføres. Vårt kurs hadde et element av dette siden vi benyttet oss av denne teknikken når vi skulle vise for eksempel, det initsielle arbeidet ved oppstarten: Lete frem systemene som editor, kompilator også videre. Jeg hevder at vi la opp til at veiledningen vår ikke skulle ha form av mesterlære, men via sin form heller stimulere til at studentene skulle lære ved egen erfaring. Når Nielsen og Kvale (1999) sier at mesterlære bruker et annet utgangspunkt enn prosjektarbeid gjør de det med: ”Prosjektarbeid lar læringen ta utgangspunkt i de nye elevenes erfaringer og motiver. I mesterlæring er det mesterens ekspertise og deltagelse i praksisfellesskapet som representerer fagets og tradisjonens krav, og som danner rammen om læringen.” Med dette utgangspunkt vil jeg nok plassere vårt eksperiment som et prosjektorientert opplegg.

Vi ville også prioritere å gjøre den enkelte aktivitet i kurset avgrenset og med et klart mål. Dette for å få effekten av suksess i noe som studentene opplevde som et vellykket eksperiment og redusere stress ved at de løste en aktivitet før de startet på

neste. Vi skulle bruke oppsummeringen for å gi studentene mulighet til å avklare eventuelle uklare punkter, samt forsterke opplevelsen av at de hadde den forståelsen vi var ute etter.

Siden vi hadde prefabrikerte sekvenser av aktiviteter kan vi definere det som undervisning, mens andre vil hevde at det er veiledning og at vi bare gav informasjon som støttet opp rundt dette. Jeg vil definere deler av det vi gjorde som undervisning. Nemlig, det at vi hadde et opplegg med ferdiglagede aktiviteter og at vi brukte tid på å beskrive en setting (kontekst) som skulle binde aktivitetene sammen. Vi hadde også orientering for studentene, før de startet på den enkelte aktivitet. Dette er å betrakte som undervisning, mens derimot er den interaksjon og diskusjon vi hadde med studentene å betrakte som veiledning. Dette siden vi la opp til at studentenes tolkinger og formuleringer skulle være retningsgivende for den verbale kommunikasjonen under veiledningen.

6.3 3 spørsmål

Mitt mål var undersøke om de tre spørsmål (4.6 Veiledning baseres på tre spørsmål) kunne bidra til at studentenes fokus endret seg fra syntaks som kunne løse problemet til at fokus ble på om de anvente de aktuelle objektorientertebegrepene på en slik måte at det kunne bidra positivit opp mot å løse problemet.

Siden jeg gjerne ville prøve ut å veilede etter de tre spørsmålene slik de er omtalt i kapittel 4.6 Veiledning baseres på tre spørsmål, og samtidig gi studentene den ”gode” følelsen, følte jeg at situasjonstilpasningen måtte bli en form for ledetråd. I dette legger jeg at å situasjonstilpasse kan forsterke studentens følelse av at veileder bryr seg om å hjelpe. På den andre side, når en skal bruke de tre spørsmålene så kan det for studenten bli en opplevelse av at veileder ikke vil svare på det problem som vedkommende formulerer. Eksempel på dette er at en gruppe i en tidlig aktivitet pekte på skjermen og sa: ”Vi er usikker, skal vi skrive: If front is clear, move?” Jeg svarte med to helt andre spørsmål: ” Hva vil dere at objektet skal gjøre? Hva er hensikten?” Dette er en situasjon som kan få enkelte studenter til å føle at de ikke får svar på det de ønsker. Dette kan komme av at studenten er mer resultatorientert enn opptatt av å forstå sammenhenger på et mer overordnet nivå som begrepsforståelse er. Intensjonen vår var å bruke kommunikasjon for å styrke selvoppfattelsen og derav få som resultat, kompetanse hos studentene. Dette skulle vi gjøre gjennom å få de til å fokusere på begrepene og at de ved å bygge en forståelse av hvorledes de kom til anvendelse i programmering, skulle anvende de slik at de fikk en opplevelse av å mestre.

6.4 Indirekte og direkte læring

Det at læreren underviser og viser eksempler er en undervisningsform som Laurillard (1993) beskriver som indirekte læring. Dette har vært den tradisjonelle akademiske undervisningsform. I akademiske miljø har ofte læring vært basert på å bygge kunnskap på andres erfaring i en akademisk kreert setting. Det hele er ofte

bearbeidet gjennom et symbolsk system av konsepter og regler. Derimot er den direkte læringen basert på å lære i direkte eksperimenter gjennom å se, høre, føle forandringer. Det hele gjøres i en setting som personene kjenner eller kan assosiere til (Laurillard, 1993). Ofte er persepsjon et viktig element i direkte læring.

Selve aktivitetene vi hadde forberedt legger helt klart opp til direkte læring idet studentene skulle se på et kart og lage en løsning som så skulle manipulere bevegelser på skjermen. Dette er å legge opp til at studentene skulle anvende persepsjon i sitt arbeid med aktivitetene. For å få til dette, la vi opp til at oppgavene skulle være korte, og det skulle være lett å se på skjermen situasjonen før og etter at løsningsforslaget ble prøvd. For at det skulle være lettere å kjenne igjen endringen, var det viktig at oppgavene ble beskrevet i en kontekst. Denne konteksten var at de skulle kjenne igjen en setting som jo er en spillsituasjon på datamaskiner. Det er en klar svakhet i vårt opplegg, nemlig at vi regnet med at alle kjente litt til spill på datamaskiner. Ikke nødvendigvis egen erfaring, men at de i hvert fall hadde sett andre gjøre det eller at de hadde sett denne type aktivitet på TV. Derimot regnet vi med at alle hadde en kulturell bakgrunn som gjorde at de i hvert fall hadde egen kjennskap til spill av forskjellig art, slike som barn bruker i oppveksten. Eksempel på dette er spill av lignede type som Ludo.

6.5 Programmeringsomgivelser

Det å ha en kontekst brukte vi også for at studentene kunne kjenne seg igjen og ha lettere for å se hva som var målet i en ny aktivitet. Med kontekst menes her kartet og programmeringen av bevegelsene til objektet på skjermen, samt at aktivitetene bygde på hverandre. Uti fra egen erfaring, hadde vi en oppfattning av at det å bytte konsept ville gjøre at det tok lengre tid å sette seg inn i en ny aktivitet. Vi oppfattet også at en samlende kontekst var viktig for å gjøre det lettere for studentene å oppfatte hvilke begrep som kom til anvendelse i hvilke situasjoner (Laurillard, 1993). Hun påpeker også at det er viktig at vi bruker vår kunnskap i en autentisk aktivitet. I vårt tilfelle betyr dette at studentene må få kunnskap om domene for anvendelse av begrepene og ikke bare begrepene for at de lettere skal kunne gjenkjenne dem.

Det kan hevdes at ved å ha større variasjon i konteksten vil studentene ha større mulighet for å se behovet for å generalisere, for der igjennom å kunne abstrahere kunnskapen (Laurillard, 1993). Vi var redd for at flere kontekster ville trekke fokus fra begrepene, og over på den til enhver tid aktuelle konteksten.

Nå brøt vi prinsippet om en kontekst når vi på slutten av kurset innførte Robocode. Dette siden programmeringen i dette systemet er mer omfattende (7.3.1 Valg av verktøy). Men vi var av den oppfatning at begge systemer hadde en robot som beveget seg i et begrenset areal, x-y planet og hadde ferdigheter som kunne programmeres. Dette skulle være så pass store likheter at det skulle gjøre det lettere for studentene å takle overgangen fra å programmere Karel J til Robocode.

6.6 Konstruktivistisk

Dette en strategi basert på en konstruktivistisk tankegang som beskrives av Jean Piaget. Selv om han selv ikke brukte dette begrepet (det ble tillagt han etter hans død), så beskriver han flere elementer som alle er en del av en konstruktivistisk tankegang. Den første er at nye ferdigheter og kunnskap er basert på hva vi allerede vet. Det vil si at vi kan ikke lære noe i et vakum. I vårt kurs brukte vi som utgangspunkt hva studentene hadde av erfaringer og kunnskap fra før. Som jeg har skrevet ovenfor hadde vi en svakhet i opplegget i det vi ikke kunne garantere at alle hadde erfaring i de forutsetningene jeg har beskrevet tidligere. På den andre side er det liten mulighet for at noen av studentene som begynner på et universitetsstudium i Norge ikke skulle ha erfaringer som vi kunne bygget på. Dersom kurset skulle bli avholdt i en annen kultur, som for eksempel på et annet kontinent, vil jeg nok anbefale at en vurderer en annen kontekst samt bruker andre begreper for å illustrere objektorienterte begreper.

Det andre elementet er at vi erfarer gjennom interaksjon med: Studenter, lærere, datamaskiner, bøker og så videre. Dette er en klar argumentasjon for vårt valg, at vi la opp til diskusjon i hele kursgruppen og at vi ønsket å benytte parprogrammering. Parprogrammering drøftes senere. Vi ønsket også at studentene skulle erfare gjennom sitt arbeid på datamaskinen. Vi la opp til at skriftlig materiale skulle være kortfattet, lett å lese i forbindelse med arbeidet på datamaskinen.

Tredje element er at vi lærer gjennom refleksjon på erfaringer. Diskusjonene og observasjonene skulle lede og motivere til at studentene skulle reflektere over det de erfarte fra aktivitetene. Det kan hevdes at aktivitetene spesielt i starten hadde for liten kompleksitet til at refleksjon kunne være et effektivt hjelpemiddel. Vi la med vilje opp til enkle problemstillinger for at studentene rimelig enkelt skulle løse oppgaven og få en god følelse av mestring, og da var det et virkemiddel at studentene hadde få problemstillinger og forholde seg til.

Fjerde element er at vi konstruerer vår egen kunnskap, det vil si at vi ikke kopierer lærerens forståelse. Vi la opp til at studentene skulle konstruere sin egen kunnskap gjennom øvelsene og diskusjoner.

Femte element er at vi lærer gjennom prosesser som bygger på det som finnes og tilpasning. I det ligger at læring bygger på den erfaring vi har, og da mener en alt hva en person har opplevd tidligere. Denne kunnskapen bruker vi for å tilpasse det som vi er i ferd med å lære. Her ser vi at læring baseres på tidligere erfaringer, noe som vi også prøvde å få til med vårt sommerkurs. Det at studentene i læresituasjonen tilpasser, betyr at vi ikke kunne være helt sikker på hvordan studentene oppfattet kunnskapen, det vil si deres opplevelse av hvordan begrepene skulle anvendes.

6.7 Behaviorism

Ormrod (2004) beskriver behaviorism som et syn på læring som en samling teorier som baseres på at det kan observeres en relativ permanent endring i atferd som et resultat av erfaring. Det vil si at læring finner sted som et resultat av en eller flere hendelser i livet til den som lærer. ”Behaviorist theories focus on the learning of tangible, observable behaviors or responses, such as tying shoes, solving a subtraction problem correctly, or complaining about a stomachache in order to stay home from school.”(Ormrod, 2004).

Som en ser av teksten er definisjonen knyttet til persepsjon. I vårt prosjekt er det nettopp persepsjon ved å utføre arbeid på et tastatur som medfører at studenten kan observere en endring i innholdet på skjermen som er et viktig virkemiddel for å oppnå læring.

6.8 Forsterkning

Forsterkning er teori som hører med til behaviorisme og Ormrod (2004) parafraserer Skinners prinsipper fra 1938 om operasjonelle betingelser slik: En respons som følges opp av noe som kan forsterke blir styrket og vil mest sannsynlig skje igjen.

Det er her verdt å merke seg at forsterkning kan være noe som gir en positiv opplevelse eller en negativ opplevelse. Det er slik at en forsterkning kan føles behagelig for en person, men for en annen kan den føles ubehagelig. Jeg skal prøve å se nærmere på hvilke elementer i vårt opplegg som kan knyttes til begrepet forsterkning slik det er beskrevet ovenfor.

Kaasbøll (2002) har en liste av elementer som kan vurderes opp mot hva som kan forsterke eller svekke læringen. Vi la opp til positiv forsterkning ved å anvende avgrensede aktiviteter, samt at settingen med spill var kjent. Andre bidrag her er at vi la opp til samme kontekst selv om studentene fikk roboten til å bevege seg i mer komplekse labyrinter utover i kurset. Det at vi brukte assosiasjon av begreper som var kjent av studentene fra før, skulle også gi en positiv opplevelse.

Selv om negativ forsterkning kan bidra positivt til læring kom det lite til anvendelse hos oss. Dette siden vi i så stor grad vektla å bruke positiv forsterkning slik jeg har beskrevet ovenfor. Den eneste negative forsterkningen jeg kan tenke at vi brukte, er at vi lot studentene selv erfare via kompilator og kartet at det de gjorde kunne være feil.
Spesielt gir nok kompilatoren en negativ opplevelse når den kan liste ut et betydelig antall feilmeldinger på grunn av et semikolon som mangler eller har kommet på feil sted. I behaverismen uttrykkes dette slik at kompilatoren gir ubehagelig stimuli når den gir feilmeldinger. Når disse opphører, så gir det opphav til en forstrekning. Denne kalles negativ fordi den består i at en ubehagelig stimuli forsvinner. En slik negativ forsterkning fremmer læring.

Både kartet og kompilator kan sees på som leverandør av en informativ forsterkning som forsterker læring. Men verktøyet Karel J som vi skulle benytte oss av, har en svakhet i at informasjonen til studentene er basert på en setting som en profesjonell programmerer opererer i. Med dette mener jeg at kunnskap om operativsystemkommandoer, editor, kompilator og feilmeldinger er mangelvare i starten. Det at det mangler, kan jo sees på som et insitament for å lære disse tingene, men vil alle studenter trives med dette i et begynnerkurs?

Ikke informativ forsterkning styrker ikke læringen og her må jeg dessverre erkjenne at operativsystemkommandoene er et eksempel på dette siden det i enkelte situasjoner ikke gis tilbakemelding om at kommandoen er utført. En illustrasjon på dette er når du kopierer en fil til en ny mappe, så får du ingen tilbakemelding om at kopieringen har gått bra. Derfor er det slik at denne type verktøy ofte krever erfaring for å tolke hendelsene på skjermen. Når våre studenter ikke hadde denne erfaringen, måtte vi kunne levere veiledning som kunne gi den nødvendige støtten på dette.

Umiddelbar forsterkning er et middel som brukes i mange læringsprosesser siden det styrker læringen. Vi la opp til å være tilgjengelig, aktiv og utdypende i vår veiledning. Det er ikke sikkert at studentene oppfattet oss helt slik, idet vi ikke alltid svarte på deres spørsmål, men at vi prøvde å bruke våre tre spørsmål. Det er også forsterkende at de umiddelbart så på skjermen dersom det var en feil som kompilatoren avdekket eller bevegelsen til objektet på skjermen. Dette gjorde at de umiddelbart kunne gjøre endringer for å rette eventuelle feil.

6.9 Parprogrammering og PBL

Parprogrammering (4.5 Parprogrammering) og PBL (4.4 Problembasert læring) har jeg tidligere beskrevet og skal her prøve å knytte begrepene opp mot vårt eksperiment.

Det at vi brukte parprogrammering gjør at veiledningssituasjonen for instruktørene blir annerledes, enn dersom vi hadde latt studentene jobbe singel. Det vi la opp til, var at veiledningen skulle ha flere nivåer. Pargruppene skulle være en form for førstelinje tjeneste, ved at studentene skulle diskutere seg imellom før de tok kontakt med en instruktør. I dette ligger det ikke at de skulle være omforent i sine problem. Men at vi ønsket at de skulle ha klargjort gjennom sin diskusjon i gruppen hva de oppfattet som et problem.

Det kan her sies at den interne diskusjonen kan gjøre at de to studentene bruker for meget tid på diskusjoner som ikke fører dem videre. Men nettopp det at vi la opp til korte arbeidsoppgaver med begrenset kompleksitet skulle bidra til at det ble lettere for deltagerne i gruppen å holde fokus oppe på de sentrale begrepene i diskusjonen. Pargruppene skulle også gi mulighet til å utforske forskjellige alternativer og måter å tenke på, ved at det kan være lettere å prøve noe i en liten gruppe på to enn i en gruppe som for eksempel er dobbelt så stor.

Men der er det å bemerke at de resultatene knyttet til parprogrammering jeg har funnet, er basert på forsøk som går over en lengre periode enn våre tre og en halv

dag. Derfor er det vanskelig å si noe om hvilken effekt på læringen av programmering det hadde at vi benyttet parprogrammering. Det vi derimot kan si noe om, er at kommunikasjonen mellom deltakerne i en gruppe bidrar til et rikere tilfang av ideer til å knytte objektorienterte begreper mot, holde deltakerne fokusert og fremdrift i jobbingen med aktiviteten (4.5 Parprogrammering).

Siden et innføringskurs i programmering ved et universitet har en del studenter, må vi påregne at enkelte av disse studentene kan ha problemer med å jobbe i grupper. Med å basere et undervisningsopplegg på problembasert læring er det overveiende sannsynlig at deler av studentmassen på et kurs opplever dette problemet. Ved University of Delaware fant de at mindre enn 10% av studentene hadde alvorlige problem med gruppearbeid. Men de har erfart at antallet av studenter som ikke foretrekker arbeid i grupper har falt signifikant over en periode på fem år (Shipman og Duch, 2001). Dette kan komme av at de som holder kurs har fått mer erfaring og gjort endringer i sitt opplegg.

Vårt opplegg med å bruke en gruppedannelse basert på parprogrammering skulle bidra til at dersom noen på kurset opplevde problem med å jobbe i gruppe, så ville en slik student ha minst mulig påvirkning på andre enn den som var i samme gruppe. Dette kan være en svakhet i det at denne ene personen blir alene og ikke kan få hjelp av andre i samme gruppe. Konsekvensen av dette var at vi som holdt kurset måtte bedrive en tettere oppfølging av den enkelte gruppe. I vårt kurs var ikke dette et problem siden antall deltakere på kurset var så pass lite. Men i kurs med 200 og kanskje opp mot 1700 må det utvikles et opplegg som kan minske konsekvensen av at en del studenter kan ha problem med gruppearbeid.

Det å anvende så små grupper som parprogrammering gir, skulle gi et bidrag til at studentene følte at det var enklere å kommunisere i en slik gruppe enn i en på opptil 8 stykker. Her kommer det inn at veilederne skulle være aktive, og kunne delta i gruppene på eget initiativ. Det kan være en vanskelig balansegang mellom veiledning og hjelp, særlig når instruktøren er aktiv. Her kommer de tre spørsmålene inn, de skulle bidra til at det ble mer veiledning enn at kursholderne gav hjelp som flytter undervisningsopplegget vekk fra PBL og mer over på forelesnings/instruktørformen. Svakheten er at når 2 stykker diskuterer så får en ikke like stor bredde i synspunkter og bidrag som for eksempel 8 stykker kan gi.

I boken “The power of problem-based learning” skriver Shipman og Duch (2001) at selv studenter som har vanskeligheter, kan lære fra gruppearbeidet. Et eksempel på dette er ”Brett”. Første kurset han deltok i ble han kraftig kritisert for ikke å ta arbeidet på alvor og ikke bidra. Ett semester senere var han mye mer positiv og han deltok fullt ut i arbeidet. ”We are pleased that Brett was able to learn how to work in groups in college rather than in a much less forgiving setting- on the job.”

7 Planlegging av eksperimentet

Vårt eksperiment som ble holdt i august 2003 var basert på et uvalg studenter som skulle begynne på Matematisk, Naturvitenskapelig fakultet ved Universitet i Oslo. Kurset gikk over 2 dager pluss at det i uken etter var en halv dag det vil si at studentene hadde 15 timer med aktivitet. Kurset var lagt opp med en målgruppe som hadde brukt PC med en tekstbehandler.

Når vi jobbet med planleggingen klargjorde jeg for meg selv, hva jeg tenkte kunne være vanskelig for en student som skal lære å programmere. Da tenkte jeg slik: Det å skrive et program innbærer en del elementer som studenten nokså raskt kommer bort i. Et eksempel er at syntaks og semantikk i programkoden, må være riktig for at et program skal fungere slikt det er tenkt. Da er det naturlig at en student fokuserer på det som vedkommende opplever som vesentlig for å produsere et resultat på en oppgave. Når syntaks og semantikk blir det som oppleves av studenten å gi feilmeldinger, vil mye av innsatsen til vedkommende lett bli fokusert mot hva som trengs for å unngå dette.

Kan dette ha påvirket de tradisjonelle innføringskurs i programmering som har vært holdt og holdes på høgskoler og universiteter? Jeg er av den formening at slik påvirkning har funnet sted, nettopp siden opplegg i slike kurs ofte baseres på å hjelpe studenten til å løse oppgaver. I dette legger jeg at undervisningsstrategien baseres på å lære studentene semantikk og syntaks, mer enn på å bygge kunnskap om for eksempel objektorienterte begreper. Derfor ble valg av undervisningsstrategier viktig for meg i diskusjonen av vårt kursopplegg.

7.1 Andre undervisningsopplegg versus min oppgave

I kapittel 5 Vurdering av bøker, omtaler jeg tre bøker som hver representerer et undervisningsopplegg. To av bøkene (5.1 Java Gently og 5.2 Java som første programmeringsspråk) er basert på å anvendes sammen med tradisjonelle forlesninger. Innføring av sentrale objektorienterte begreper skjer på et for sent tidspunkt i progresjonen som er lagt opp og dette er heller ikke i overensstemmelse med min hypotese (1.2 Problemstilling). Den tredje boken (5.3 Java BlueJ) innfører og anvender objektorienterte begreper på et tidlig stadium i stoffet. Boken legger opp til at studenten skal bygge kunnskap ved praktiske øvelser på datamaskinen. Så hvorfor anvendte vi ikke denne eller en tilsvarende bok i vårt kurs når vi også baserte oss på å bygge kunnskap ved praktiske øvelser på datamaskinen?

Jeg hevder at ved å anvende en bok, medfører dette at begreper og øvelser lett følger bokens begrepsapparat. Siden boken er laget for å kunne bygge kunnskap etter selvstudium prinsippet er alle beskrivelser av begreper og øvelser tilpasset dette.

Som en konsekvens av dette og det faktum en bok bygger begrepsapparatet på forfatternes kunnskap og erfaring (2.4 Erfaringsbakgrunn), behøver boken slett ikke

å være i overensstemmelse med studentens erfaring. Vi ønsket å knytte begrepene vi jobbet med opp mot begreper som studentene kjente fra før eller som vi kunne omforenes på.

Vi baserte oss på at studentene skulle delta i diskusjoner som vi skulle bruke for å knytte begrepene sammen, og stimulere til samarbeid innad i gruppen. Dette, og at vi ville gi studentene mulighet for å utforske sine egne oppfattninger mot andre sin oppfattning, hadde som forutsetning at vi kunne situasjonstilpasse opplegget vårt. Jeg finner at vårt ønske om fleksibilitet i opplegg vanskelig kan tilpasses innholdet i en bok av lignende karakter som de som er omtalt i kapittel 5 Vurdering av bøker.

Videre ønsket vi at opplegget skulle være en undervisningssituasjon, mer enn et selvstudium basert på en bok. Vi ønsket å gå lenger i å bygge kunnskap ved øvelser enn boken Java BlueJ gjør, samt at jeg ønsket å teste ut om øvelser som støttes av veiledning kan bidra til bygging av objektorienterte begreper.

Vi fant ut at med vårt perspektiv ville boken Java BlueJ være lite egnet. Vi fant heller ingen annen bok som tilfredsstilte våre ønsker, derfor valgte vi å lage et eget opplegg for eksperimentet.

7.2 Valg av undervisningsstrategier

Som tidligere omtalt er det i COOL en gruppe mennesker som er interessert i didaktikken knyttet til det å lære programmering etter objektorienterte prinsipper. Denne gruppen hadde flere møter hvor de blant annet ble delt opp i forskjellige interesseområder og laget forslag til aktuelle eksperimenter. Det ble i det vesentlige arbeidet med to forslag. Et basert på BlueJ og Resturant eksemplet (COOL) til Kristen Nygård. Det andre basert på Karel J og en robot som beveger seg i et todimensjonalt kvadratur. Det ble bestemt at vårt eksperiment skulle baseres på Karel J og lages som et sommerkurs for studenter. Kurset skulle være et introduksjonskurs til begynnerkurset INF1000 (Begynnerkurs i objektorientert programmering) på Institutt for Informatikk ved Universitetet i Oslo. INF1000 er også et kurs som tas av alle studenter ved Matematisk Naturvitenskapelig fakultet. Vi ønsket å legge opp til at studentene ikke hadde noen erfaring med programmering fra tidligere. Så langt i arbeidet hadde Richard Borge jobbet mest med Karel J og det var derfor naturlig at han var hovedansvarlig for Karel J. Den didaktikken vi skulle anvende i kurset ble utarbeidet av Richard Borge og meg selv i nært samarbeid med Fjuk, Groven og Kaasbøll. Det ble lagt opp til at Richard og jeg skulle lage materialet som skulle anvendes samt være i undervisningssituasjonen i kursgjennomføringen. Vi skulle bruke de andre tre som konsulenter i arbeidet.

Øvelsene som vi utarbeidet ble laget ut i fra en liten ideskisse og bearbeidet på våren og forsommeren 2003.

Settingen som kurset skulle gjennomføres i var et klasserom med 15 PC'er. Rommet hadde tavle og overheadprojektor. Det ble også bestemt at vi ønsket å anvende Intermedia ved Universitetet i Oslo sin kunnskap om å dokumentere undersøkelser med video opptak.

Det ble tidlig klart at vi skulle ha noen hovedfokus i kurset:

- Begrepsdannelse
- bruk av grafisk grensesnitt (første hypotese)
- bruke en meget kort skriftlig dokumentasjon
- veiledningen skulle baseres på tre spørsmål (som går på innsikt).

Disse hovedfokus medførte at vi under det forberedne arbeidet kom frem til at vi trengte en del kriterier. Disse kriteriene skulle brukes for å bygge opp om hovedfokus for kurset:

- Fokusere på begrepsdannelse
- skriftlig dokumentasjonen skulle være meget kort
- øvelsene skulle baseres på erfaring fra tidligere
- hver aktivitet skulle bare tilføre ett nytt problem
- veiledning under aktiviteter baseres på tre spørsmål (som går på innsikt)
- innledning/oppsummering skulle baseres på interaksjon
- parprogrammering.

Jeg ble under diskusjonen klar over at flere av punktene ovenfor kan komme til anvendelse ved undrevisningsmetoden som beskrives som problembasert læring. Selv om vi på dette tidspunkt ikke hadde problembasert læring (4.4 Problembasert læring) med på listen over fokuserte emner. Det var først når planleggingen av undervisningen var helt i sluttfasen at vi definerte at denne metoden sammen med parprogrammering (4.5 Parprogrammering) var sentral for gjennomføringen av vårt eksperiment.

7.2.1 Fokusere på begrepsdannelse

Vi skulle prøve å lage et opplegg som fokuserte på begreper som vi oppfattet som sentrale for studenter som skulle ta sitt første programmeringskurs. Det var også klart at vi i løpet av det korte kurset skulle prøve å bygge en innsikt om sentrale begreper innen objektorientert programvareutvikling. Vi ønsket å ha en arbeidssituasjon som hadde likhetstrekk med arbeidssituasjonen en programmerer opplever. Det vil si at vi ønsket at studentene skulle skrive program i en eller annen editor for så å kompilere og eventuelt rette i programmet etter at de hadde fått feilmeldinger. Så skulle de starte kjøringen av programmet og se hva som skjedde på kartet. Denne interaksjonen er jo en arbeidsform som er virkeligheten for personer som skriver programmer som skal kjøres på en datamaskin. Vi ønsket at studentene skulle bli kjent med interaksjonen og en del begreper som anvendes av programmerere i deres arbeid. Typiske begreper er: Objekt, klasse, metode, data og kompilere.

Når det gjelder begreper knyttet til selve programmene, ville vi prøve å innføre et begrep om gangen og gjennom å bygge en viss forståelse for begrepet, bruke denne kunnskapen til å innføre nye begreper.

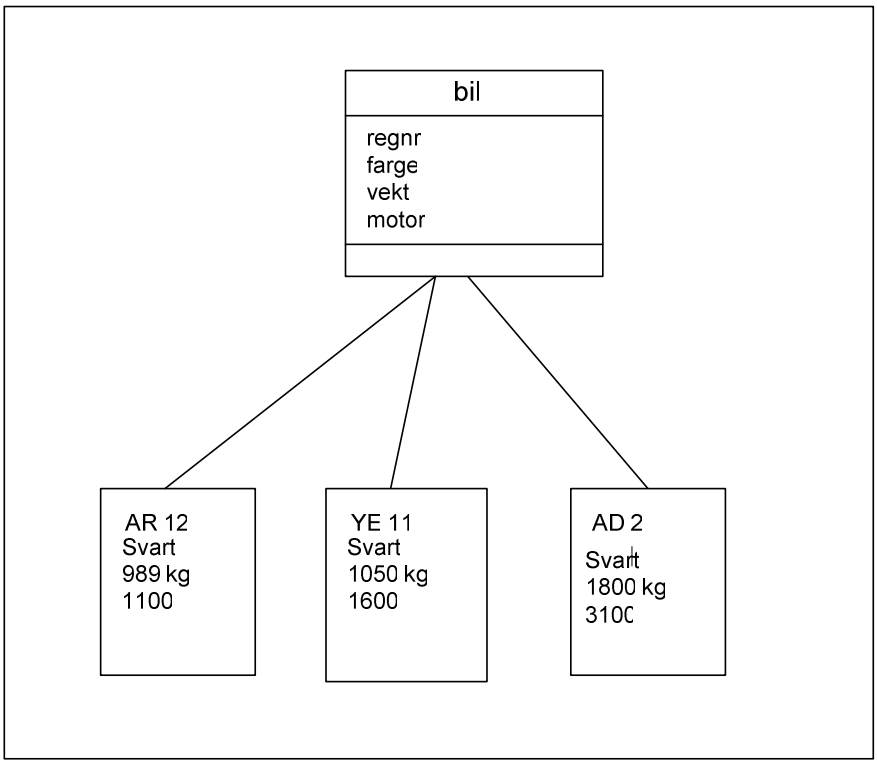
Rekkefølgen som vi skulle presentere begrepene i, ble bestemt ut i fra hva vi oppfattet som best egnet for å bygge kunnskap på tidligere erfaring hos studentene. Logikken er konstruktivistisk (6.6 Konstruktivistisk) i det den legger opp til å bygge på det som tidligere er erfart.

I hver øvelse skulle vi basere oss på at kunnskapen skulle bygges via en dialog som tok utgangspunkt i studentenes erfaringer. Dette medførte at vi skulle bruke deres erfaring med begreper som utgangspunkt for samtalene. Det første begrepet vi skulle innføre var objekt og knytte dette til hva de var vant til å bruke ordet om. I aktivitetene ønsket vi at studentene skulle kunne observere at objektene gjorde noe som de ble instruert til fra et program. Siden vi ønsket at studentene skulle bygge på egen kunnskap, måtte vi finne noe som vi regnet med de hadde litt kjennskap til. Men vi måtte passe oss for å legge inn forutsetninger som gjorde det vanskelig å delta i diskusjon og arbeid, dersom de kunne svært lite om bruken av datamaskiner.

For å illustrere hvorledes vi ønsket å innføre begreper bruker jeg innføring av objekt og klasser. Siden studentene allerede i første aktivitet skulle se et objekt og skulle ha det til å bevege seg, var objekt det første begrepet vi valgte å innføre. Da skulle strategien være at vi skulle starte med å spørre om hva de tror et objekt er. Tilbakemeldingene skulle vi bruke som grunnlag for å snakke om alternativene til forståelse av begrepet. Vi skulle her ikke prøve å komme frem til en presis definisjon av ordet, men mer prøve å skape en intuitiv forståelse hos studenten. Vi skulle legge opp til at de etter hvert i kurset skulle få en klarere forståelse på hvorledes ordet objekt ble brukt i forbindelse med programmering. Den strategien med å bruke studentenes erfaringer for å innføre nye begrep, skulle vi prøve å følge i hele kurset. Vi kunne ikke helt vite hvor aktiv studentenes skulle bli, så vi hadde noen alternativ som vi kunne presentere. Vi ønsket også å være aktive og bruke muligheter for å initiere diskusjon når vi følte behov for dette.

Det neste steget var å legge opp til å bruke den forståelsen de hadde av ordet objekt til å innføre ordet klasse. Klasse er ”Objects are created from **classes**. The class describes the kind of object; the objects represent individual instantiations of the class.” (Barnes og Kölling, 2003)

Vi ønsket å starte med en diskusjon om ordet klasse, hvor jeg etter hvert skulle innføre en forklaring om at en klasse i programmering brukes som et begrep på objekter som har samme konstruksjon. I en oppfølging på dette skulle vi bruke en enkel figur. Studentene skulle få en forklaring på at figuren ikke er helt korrekt og at de senere kom til å lære mer nøyaktig notasjon. Figuren skulle bare brukes som en skisseillustrasjon.



Figur 17: Illustrasjon av klassebeskrivelse med objekter

Jeg gjør oppmerksom på at jeg brukte betegnelsen kg i forbindelse med vekt, men ikke betegnelse på størrelsen på motor. Jeg har ingen god forklaring på denne inkonsistens i anvendelse av betegnelser.

Så skulle vi presentere en forklaring om at en klasse brukes for å opprette et objekt. Her skulle jeg henvise til figuren og forklare at forskjellen på objektene kan være fargen, registreringsnummer eller andre parameter, men objekter fra samme klasse har de samme type data.

7.2.2 Skriftlig dokumentasjon skulle være kort

Vi ønsket å bruke muntlig kommunikasjon som et sentralt element i vår kommunikasjon med studentene. Derfor skulle vi prøve å legge opp til at det skriftlige materialet som skulle leveres ut, var meget kort. Men det måtte også gi mulighet for å kunne lese litt bakgrunnsstoff. Vi gjorde dette ved å legge opp til at dokumentasjonen skulle bestå av to nivåer. Et nivå som studentene skulle kunne lese dersom de hadde interesse av å vite mer og utdype med bakgrunnsstoff. Det andre nivået skulle inneholde bare det som var nødvendig for å jobbe med oppgavene, så da studentene møtte opp skulle de få utlevert:

- Introduksjon (2 sider) Richard Borge (Vedlegg 1)
- Karel J, en aldri så liten innføring i robotkommunikasjon. (5 sider,Vedlegg 1)

Den siste inneholdt også den første aktiviteten som studentene skulle programmere.

Vi skulle prøve å legge opp til en uformell stil og aksept for uformel bruk av begreper. Dette skulle vi prøve å gjenspeile i den skriftlige dokumentasjonen. Teksten skulle fokusere på de vesentlige begreper og mer fungere som en huskeliste. Denne måten å bruke dokumentasjonen er beskrevet av Herskin i 1994, dette er omtalt i 2.6 Forelesning.

7.2.3 Aktiviteter skulle baseres på erfaring

Vi ønsket å lage aktiviteter som bygget på begreper som studentene kunne kjenne igjen. Derfor var det viktig at første aktivitet prøvde å innføre et hovedbegrep som studentene kunne kjenne fra før. Vi valgte som tidligere nevnt objekt og skulle gi det en entydig identitet ved å navngi det Karel. Her er det å merke seg at vi så det som lite motiverende å se et objekt som ble stillestående på skjermen. Derfor skulle neste aktivitet få objektet til å flytte seg et skritt frem ved å gi en kommando. Kommandoen er ”*move*” som betyr: Beveg objektet et skritt frem. Vi skulle snakke om objekt og identitet for å bygge en forståelse for begrepene, da de skulle brukes som fundament for å innføre nye begreper.

Vi ønsket ikke bare at studentene skulle kunne bygge på sine programmeringserfaringer, men også at studentene skulle erfare at gjenbruk kunne være et effektivt hjelpemiddel. Dette skulle vi gjøre gjennom at en modul som var programmert i en tidligere aktivitet kanskje kunne benyttes i en senere aktivitet. Koden kunne brukes slik den var eller bare litt modifisert.

7.2.4 Hver aktivitet skulle tilføre et nytt problem

Aktivitetene skulle bygges opp slik at studentene bare skulle få presentert et nytt moment som skulle læres i hver aktivitet. Vi brukte her som utgangspunkt at det ville være lettere for studentene å fokusere på ett moment, fremfor flere momenter på en gang.

Med dette utgangspunktet ble de et hovedpunkt hos oss at studentene frem til et visst punkt skulle oppleve at de bare hadde ett problem å forholde seg til i hver aktivitet. De skulle ha den tryggheten at dersom de kom borti noe som de opplevde som to problem, var det ene noe de hadde hatt før. Samtidig skulle vi prøve å lage progresjonen i oppgavene slik at de skulle ha nok erfaring til å være klar over hva som var kjent og hva som var det nye problemet.

Siden vi hadde en meget kort introduksjon til hver aktivitet ville det bli vanskelig å få denne fullstendig utdypende. Derfor måtte vi informere studentene om at dersom de hadde jobbet med å definere et forslag til løsning, fortsatt hadde mer enn ett problem, burde de overveie om de jobbet med riktig løsning eller om det kunne være andre måter å løse det på. Dette så vi som en utfordring for det veiledningsarbeidet vi skulle gjøre, spesielt siden dette var et argument for å gjøre veiledningen situasjonsavhengig.

7.2.5 Veiledning

Veiledningen jeg ønsket å benytte i eksperimentet var basert på to elementer. Det første var de tre spørsmål (4.6 Veiledning baseres på tre spørsmål), og det andre var at den i stor grad skulle være situasjonsbetinget.

De tre spørsmål

For å stimulere til at studentene jobbet med det de måtte oppleve som problem eller vanskelig, fremfor å søke lettvinte løsninger, skulle vi stimulere deres egen kritiske tenking. Vi skulle unngå å fortelle løsningen eller selv hinte om den. Som jeg tidligere har beskrevet i kapittel 4.6 Veiledning baseres på tre spørsmål, hadde jeg lest en artikkel av Schoenfeld (1992) hvor han fokuserer på å bygge innsikt i et tema. I arbeidet med øvelsene bruker han små grupper på tre eller fire studenter hvor han går rundt og veileder. Han reserverer seg retten til å kunne stille tre spørsmål når som helst: Hva gjør dere?, hvorfor gjør dere det?, og hvordan hjelper det dere?

Disse spørsmålene skulle vi bruke til å knytte fokus på begrepene som var behandlet så langt. Siden vi regnet med at alle kunne norsk, ville vi stille spørsmålene på norsk. Vi ville også situasjonstilpasse formuleringene, men ville i størst mulig grad prøve å holde oss til å snakke om hvorledes studentene anvender begrepene som er temaet i aktiviteten og hvorledes de oppfatter dem.

Det første spørsmålet skulle vi bruke for å få studentene til å beskrive hva de tror koden instruerer maskinen til å gjøre. Dette inkluderer også å komme inn på hvordan de enkelte kodebiter henger sammen. Det andre spørsmålet skulle vi bruke for å få studentene til å klargjøre hva de skulle bruke de enkelte delene til i forhold til problemstillingen de skulle jobbe med. Det tredje spørsmålet skulle brukes til å få studentene til å overveie konsekvensen av det som ble foreslått.

Situasjonsbetinget

For å få til en dialog som studentene kunne oppfatte som motiverende, og hjelpe på utvikling av begrepsforståelse, er det viktig å ha en kommunikasjon som kan bygge på en felles begrepsoppfattelse. Språket som vi anvender er et viktig hjelpemiddel for å oppnå dette. Som jeg har beskrevet i kapittel 2.3 Språk, så kan ulik erfaring som etnisk bakgrunn, oppvekst og utdanning gi forskjellig begrepsapparat (2.4 Erfaringsbakgrunn). Det er derfor viktig at veiledningen tar hensyn til dette ved å situasjonstilpasse hvilke begreper som forutsettes kjent og som anvendes i språket. Jeg omtaler også i kapittelet forskjellen på muntlig og skriftlig språk. Nettopp for å møte studentene på deres oppfattelse, kan det i noen sammenhenger være nyttig å anvende formuleringer som i skriftlig sammenheng ikke vil være godtatt. Da tenker jeg på at en bruker beskrivelser eller erstatningsord som for eksempel objekt er lik ting eller mal/tegning er lik klasse.

Situasjonsbetinget veiledning innebærer også at veiledningen i stor grad tar hensyn til hvor langt studenten eller gruppen har kommet i sin modenhet i å anvende begreper som tidligere er behandlet. For eksempel at en gruppe anvender lite modularisering i programkoden, mens det i kursopplegget legges opp til at det bør brukes. Da vil jeg ikke at vi skal dra fokus vekk fra det som er hovedtema i problemet det jobbes med. Veileder kan antyde at en annen struktur kunne ha bidratt til en enklere jobbsituasjon for studentene.

7.2.6 Studentdrevet progresjon

På et studium ved høgskole/universitet har det vært kutyme at den som legger opp kurset gjør dette ved å lage en plan for undervisningen som i hovedsak styres av litteratur som skal anvendes (2.5 Bøker og Kölling (2005)). Generelt kan det vel sies at forelesningene (2.6 Forelesning) kan oppleves som meget styrende av studentene med hensyn på progresjonen i det aktuelle kurset. Det er også slik at denne progresjonen ofte er planlagt ut i fra forelesers bakgrunn og erfaring, som ofte er en annen enn studentene har (2.4 Erfaringsbakgrunn). I dette følger det at studentens behov og opplevelse får en lavere prioritet i forhold til fremdriftsplanen. Jeg ville bruke en annen innfallsvinkel idet jeg valgte å prioritere at studentene skal føle at de under kurset når de ønskede kunnskapsmålene (Hofset, 1995) og har en god opplevelse (4.2 Mestring).
Jeg la dermed opp til at det kunne bli slik at vi måtte droppe aktiviteter på grunn av begrensning i disponibel tid. I dette ligger det at vi ønsket å prøve ut om det var mulig å få den ønskede progresjonen med denne innfallsvinkel.

Det kan hevdes at dette er en gal prioritering ut i fra at et studium skal ha som primærmål å bygge faglig kunnskap og innsikt, og progresjonen derfor må være førende for planleggingen. I boken ”The power of problem-based learning”(redaktører: Duch, Groh og Allen, 2001), fremføres det at formulering av målene i oppgavene kombinert med veiledningen kan brukes for å få den ønskede progresjon i læringen. Det argumenteres også for at det å dele opp et mål i flere delmål kan være et virkemiddel.

Vårt opplegg skulle ha som hovedmål at studentene skulle lære å skrive program som kunne få en robot til å bevege seg på skjermen. Delmålene skulle være klart avgrensede aktiviteter som studentene kunne knytte kunnskap til, bygget på situasjoner de kjenner fra før. Det kan hevdes at hver av arbeidsoppgavene ble for snevre til å illustrere kompleksiteten og sammenhenger. Nettopp at arbeidsoppgavene var enkle å få oversikt over, skulle være et bidrag til at vi lettere kunne holde kontroll med gruppenes fremdrift.

For å få studentene til å holde interessen for å jobbe aktivt og lære mer hadde vi en del tiltak som vi ville prøve å bruke. Disse er omtalt som 7.2.1 Fokusere på begrepsdannelse, 7.2.2 Skriftlig dokumentasjon skulle være kort, 7.2.3 Aktiviteter skulle baseres på erfaring, 7.2.4 Hver aktivitet skulle bare tilføre et problem, 7.2.5 veiledning (de tre spørsmål og situasjonsbetinget), 7.2.7 Innledning/oppsummering baseres på interaksjon og 7.2.8 Parprogrammering. Jeg gjør oppmerksom på at

tiltakene 7.2.3 Aktiviteter skulle baseres på erfaring og 7.2.4 Hver aktivitet skulle bare tilføre et problem, utdypes nærmere i kapittel 4.4 Problembasert læring.

Studentdrevet progresjon kan gjøre at enkelte studenter ikke får nok progresjon for å nå kunnskapsmålene. Dette kan komme av at de mister motivasjonen eller bruker for lang tid på hver aktivitet.

Her skulle veiledningen bidra til å lede arbeidet slik at studentene fikk mestringsfølelse og kanskje gi ekstra utfordring dersom studentene ble tidlig ferdig med en aktivitet. Dersom det ble slik at en gruppe hang etter eller hadde større vanskeligheter enn de andre, kunne også dette lettere utjevnes med at hver aktivitet var så kort at det ikke ble noe som vi mente kunne være en merkbar ventetid for de andre studentene.

Vår progresjon skulle styres av at alle grupper skulle ha laget en løsning på et problem før vi presenterte neste problem. Dette for at alle skulle føle at de starter på neste problem med samme forutsetninger. Som hjelpemiddel for å sjekke om kunnskapsmålene for hver aktivitet var oppfylt ville jeg bruke innledning og oppsummering (7.2.7 Innledning/oppsummering baseres på interaksjon). Derfor var det viktig at vi aktiviserte flest mulig i dialogen.

7.2.7 Innledning/oppsummering baseres på interaksjon

Med bakgrunn i at den skriftlige dokumentasjonen som skulle deles ut ved hver aktivitet skulle være meget kort, fikk vi et argument for å anvende en situasjonsbetinget kommunikasjon med studentene. I dette ligger det at vi i innledningen skulle basere oss på studentenes tidligere erfaringer og prøve å knytte det nye problemet opp mot stoff som vi tidligere hadde hatt oppe i samtalene. Men vi ville også gi studentene rikelig mulighet til å prøve ut sine ideer og eventuelle spørsmål i klassen. Kommunikasjonen skulle også bidra til at studentene hadde en følelse av at veilederne var oppdatert på hva studentene oppfattet som vanskelig.

I starten av kurset skulle vi bruke innledningene til å stimulere at studentene fikk en opplevelse av at de hadde behov for den aktuelle type kunnskap i arbeidet med aktiviteten. Som jeg har beskrevet i kapittel 4.4 Problembasert læring, er dette for at det skulle være lettere å lære av informasjonen og de ferdigheter studentene skulle jobbe med i aktiviteten. (Burch, 2001). Samtidig ville vi stimulere at de erfarte hvordan mye av den kunnskap som er nødvendig i arbeidet med aktivitetene kan bygges på tidligere kunnskap (4.4 problembasert læring, Duch, på hjemmesiden til UoD). Ved å ha en muntlig innledning hvor vi la opp til å prøve og få med flest mulig i diskusjonen, skulle vi prøve å inkludere alle i et samarbeid i gruppen. Dette for gjøre det lettere for deltagerne å bidra i grupper senere i studiet.

Interaksjonen ved innledning og oppsummering skulle brukes som et sentralt virkemiddel for å hjelpe til med fokusering på objektorienterte begreper. Dette var av betydning for oss, siden det er lettere å starte med det objektorienterte paradigmet enn på et senere tidspunkt å gå over fra det prosedurale til det objektorienterte paradigmet.

Kommunikasjonen skulle bidra til at vi fikk en positiv forsterkning av læringen og at følelsen av mestring ble mer bevisst for studentene.

7.2.8 Parprogrammering

Begrunnelsen vår for å satse på parprogrammering (4.5 Parprogrammering) var flere. Vi hadde et kort kurs og ønsket å fokusere på begreper og bidra til at selve settingen skulle dempe ned fokus fra kode og problemer rundt dette, til de begrepene vi ønsket å bringe i fokus. Ved å sette deltagerne sammen to og to kunne de utfylle hverandre når det gjaldt selve programmeringen og tolkingen av det de skulle se på skjermen. Det var også et ønske at de skulle kunne diskutere og fremføre sine egne erfaringer ovenfor hverandre i konkrete situasjoner de kom til å oppleve under arbeidet med aktivitetene.

Vi håpet også at det skulle bli lettere å veilede dersom det var to personer å henvende seg til. Dette med bakgrunn i at vi håpet at dersom den ene ikke husket, så kunne forhåpentligvis den andre huske noe når vi skulle ha en dialog med studentene under arbeidet med aktivitetene.

Dette ville være spesielt viktig, ettersom kurset var laget for studenter uten noen erfaring i programmering og at de skulle jobbe med aktiviteter i en setting som var ukjent for dem. Det på et tidspunkt å huske alt som tidligere har vært tatt opp i kurset, samt anvende dette i en ny situasjon, forutsatte vi ville være enklere i en setting med to og to som arbeidet sammen. Vi skulle også prøve å oppfordre dem til å samarbeide med andre grupper når de følte behov for dette.

Vi var også interessert i å undersøke om konseptet vi skulle prøve, ville ha en fordel av at studentene jobbet sammen i par. Det å ha noen å diskutere med, ville forhåpentlig føre til at studentene lettere kom frem til en forståelse av begrepene og hvorledes de henger sammen.

7.3 Plan for kursgjennomføring

7.3.1 Valg av verktøy

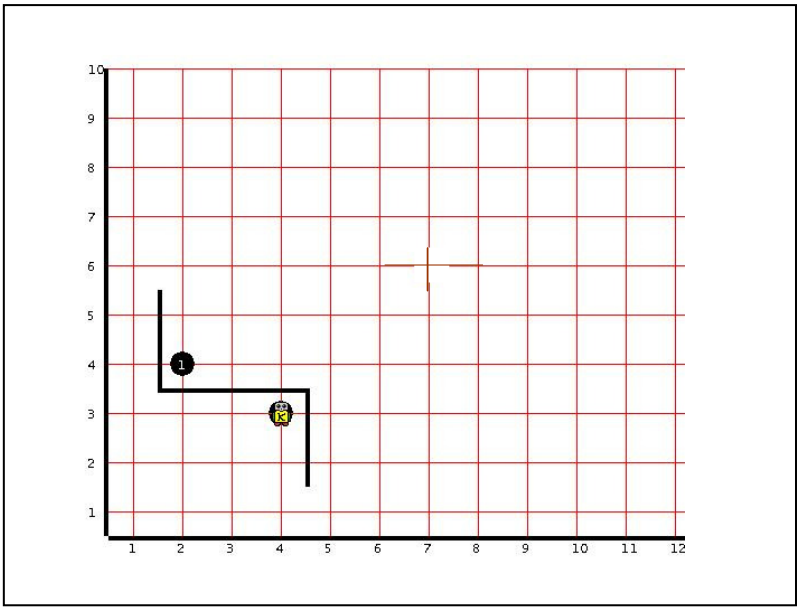
Her ønsket vi å anvende et verktøy som var enkelt å lage instrukser i og som i bruk hadde likhetstrekk med arbeidssituasjonen som en programmerer jobber i. Vi skulle prøve å introdusere en del teknikker som kan være lurt å anvende for å lage program som kan være modulære og godt dokumentert, samt teknikker for å finne feil og teste programdeler. Vi bestemte oss for at de arbeidsmetodene vi ønsket å introdusere ikke måtte overskygge vårt hovedfokus. Det vil si at vi skulle knytte kommentarer om teknikker til eksempler som deltagerne opplevde i sitt arbeid med aktivitetene. For å illustrere dette: ”Istedenfor å skrive så mye kode at det blir litt

vanskelig å holde oversikt, kan det være lurt å skrive en liten modul for så å få den til å virke før dere går videre."(fra kurset).

Vi hadde i en tidlig fase fokusert på to verktøy BlueJ og Karel J, som begge er laget for å skrive program i programmeringsspråket Java. Siden begynnerkurset i programmering ved Oslo Universitet er basert på å bruke programmeringsspråket Java ville det være en klar fordel å anvende samme språk. Dette med bakgrunn i at vi skulle ha et kort sommerkurs for et lite antall studenter og ønsket at studentene fikk en enklest mulig overgang til begynnerkurset.

Karel J

Vi fant at BlueJ hadde en for komplisert verden for vårt korte kurs. Det ville ta for lang tid å komme til et nivå som vi ønsket. I tillegg var det slik at et annet eksperiment innfor COOL prosjektet skulle anvende BlueJ, og det var ønske om å prøve flere verktøy. Vårt valg falt på Karel J. Dette verktøyet har en enkel grafisk fremstilling av en verden som roboter av typen Karel kan bevege seg i. Verden som anvendes er bygget opp av en matrise som er todimensjonal og hvor det kan legges ut vegger. Roboten kan bare bevege seg etter matrisemønstret. Et enkelt eksempel:



Figur 18: En enkel verden.

Posisjoner i denne verden spesifiseres med to tallverdier, for eksempel finnes en sort skive i posisjon 4,2. Verden kan bygges ut med å plassere vegger og det kan plasseres flere roboter inn i verden. Karel J er bygget opp slik at når studenten skriver et program kan vedkommende bruke en del predefinerte egenskaper. Disse egenskapene kan kombineres og bygges sammen til nye egenskaper.

```
// Snu til høyre
void turnRight() {

   turnLeft();
   turnLeft();
   turnLeft();
}
```

Figur 19: Hvordan bygge funksjoner.

Denne viser hvorledes en får roboten til å snu seg 90 grader til høyre, ved å utføre en snu til venstre 90 grader tre ganger.

Robocode

Vi var av den formening at Karel J kunne gi en grei setting som har stor likhet med det som en programmerer opplever i en jobbsituasjon, og var fin å starte med. Men vi ønsket å prøve et verktøy som for studenten kan oppleves som mer avansert. Valget falt på Robocode. Dette fordi at vi ønsket å bruke dette som noe å ”strekke seg etter” og som ”rosin i pølsen” på slutten av kurset.

Robocode er utviklet av Mathew Nelson ved IBM Alphaworks. Konseptet bak Robocode er at det er et spill hvor brukeren skal skrive et Java program som skal gi egenskaper til en robot. Denne roboten er en tanks og befinner seg i et skjermbilde som definerer Robocode verden. Tanksen skal skyte i stykker andre tankser, som befinner seg i samme verden. Det er verdt å merke seg at programmereren bare kan programmere egenskapene til tanksen, ikke styre den under spillet. Spillet er et spill som inneholder mye action og farten roboten beveger seg med er så pass stor, at det kan være vanskelig å følge med på hva som skjer.

Dersom dette systemet skal anvendes som innføring i programmering, vil det høyst sannsynlig ta lang tid før vedkommende kan observere at roboten gjør noe på skjermen. I dette ligger det at det tar lang tid før vedkommende får til å skrive så mye kode som må til for at en kan observere på skjermbildet at roboten utfører noen egenskaper. Det å skrive kode krever med andre ord litt erfaring og det er mye å sette seg inn i.

7.3.2 Antall på kurset

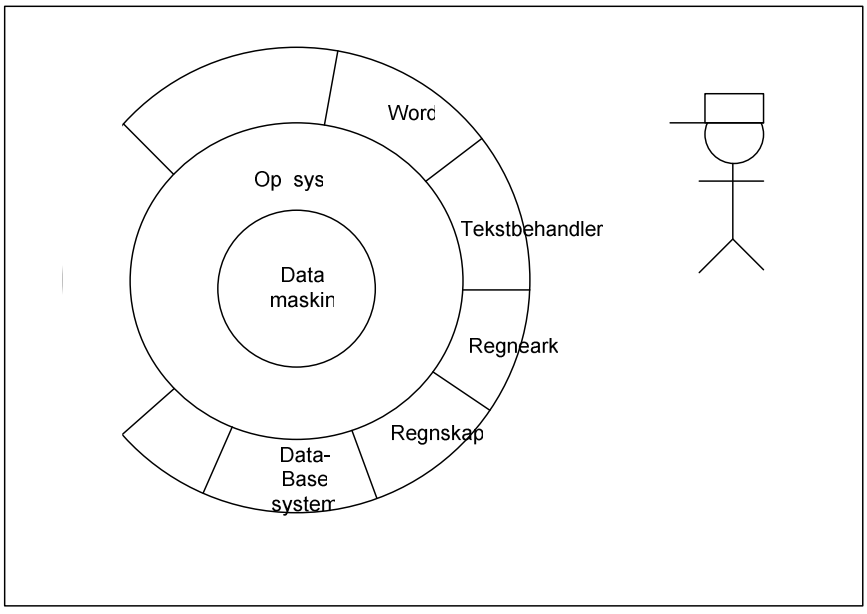
Vi avgjorde i våre diskusjoner under forberedelsene at inntil 25-30 stykker ville være et bra antall. Dette mente vi var håndterbart for oss, siden vi var to instruktører. Pluss at vi kunne påregne noe støtte fra observatørene ved spesielle

behov. Dette tallet ble også valgt ut i fra at det var enkelt å skaffe et rom som egnet seg til formålet. Vi ønsket ikke at andre studenter skulle være i samme rom, idet vi regnet med at det kunne virke forstyrrende og kanskje hemme dialogen som vi la opp til. Dessuten ville vi ha et antall som vi følte gjorde det greit å holde kontakten med alle studentene. Det kan fremføres at for enkelte studenter vil nok en gruppe på 25-30 bli for mange til at de føler seg komfortabel til å eksponere seg selv og føle at de er deltagende i diskusjonen i plenum.
En annen grunn til antallet var at vi ønsket å benytte oss av parprogrammering og dette ville da gi ca 12-15 grupper, noe vi mente ville være rimelig oversiktlig å holde kontakten med for 2 instruktører.

7.3.3 Starten på kurset

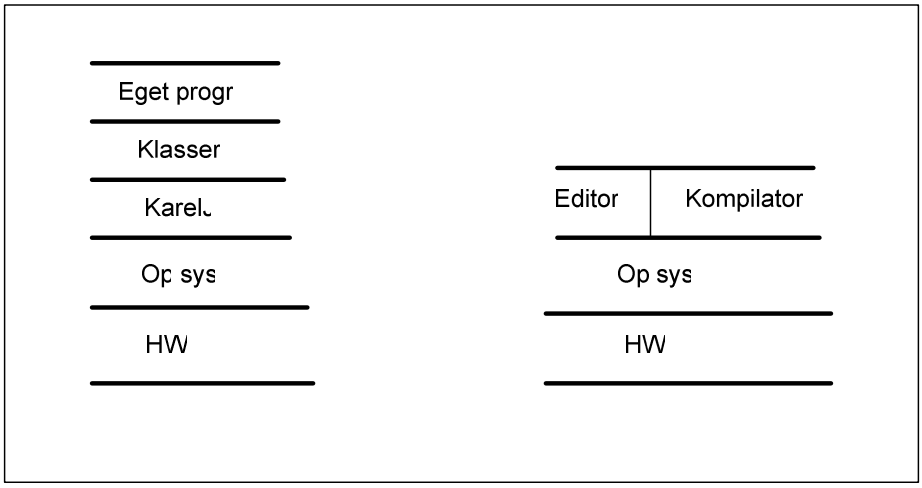
Siden deltakerne var studenter som var nye ved universitetet skulle de få hjelp til å logge på nettet. Vi planla at vi via en projektor skulle vise hvorledes de skulle starte en enkel tekstbehandler for å skrive programmer i. I tillegg skulle de anvende verktøy for å kompilere programmene sine. Planen var å vise via projektor, og gi støtte til de som trengte dette for å få etablert arbeidsbordet på skjermen. Dette regnet vi med skulle gå greit å få etablert. Under dette arbeidet skulle vi benytte sjansen til å innlede til muntlig kommunikasjon. Vi la opp til å bruke enkle begreper som brukes i daglig tale som forespørsel, beskjed og svar, for så etter hvert å innføre begreper som brukes i arbeidet med utvikling av programmer.

Så skulle vi fortelle litt om at datamaskinen ikke gjorde noe på egen hånd fordi maskinvaren ikke har noen intelligens, men at maskinvaren måtte få instruksjoner i fra et program. Siden vi regnet med at studentene hadde brukt datamaskiner og var kjent med brukerprogram som tekstbehandler og regneark skulle vi bruke en skisse på tavlen som beskrev en lagdeling.



Figur 20: Lagdeling med sirkel

Vi bestemte oss for å bruke sirkel fordi vi ønsket å understreke at for komme i kontakt med maskinvaren, må en benytte program og at flere program kan være på samme nivå. Vi kan ved hjelp av modellen vise at det er mulig å kommunisere direkte med operativsystemet. Et sentralt punkt som vi ønsket å få frem er at programmene sender beskjeder til hverandre og får svar. Vi ville også tidlig klart skille mellom data og program, så dette skulle vi kommentere kort. Siden vi ønsket å vise at lagdeling i programvare er mye brukt i dataprogram skulle vi vise alternativ modell for å illustrere dette:



Figur 21: Illustrasjon på kommunikasjon mellom lag.

Denne måten å illustrere på skulle vi bruke for å illustrere under diskusjoner og når vi skulle innføre nye begreper. Vi skulle presisere at modellen ikke er fullstendig og at hvert lag kan deles i flere lag.

Etter denne skisselignende innledningen om datamaskinens virkemåte skulle vi fortelle om Karel og hans verden. Vi ville starte med å fortelle om en verden hvor det var objekter. For at vi skulle få litt mer erfaring på hvilke erfaringer studentene satt inne med, ønsket vi at de skulle beskrive hva de trodde objekter var. Ut i fra dette skulle det være en dialog om begrepet. Det å opprette et objekt skulle vi beskrive som en bestilling til en fabrikk som leverte objekter. På forhånd skulle vi lage et program som opprettet en verden og plasserte et objekt inn i denne.

Med bakgrunn i: ”Students don’t have to be exposed to all of the perhaps messy details of a complex language initially.” (Bergin, 2000) laget vi en mal som studentene skulle bruke som utgangspunkt for det videre arbeidet med å produsere kode:

```
// Lager egne pakker for bedre oversikt
    package karelTheRobot;


    /*
     * Her er innpakningen til programmet. Husk å bytte ut navnet
     * under fra ”DinFil” til noe annet.
     */
    class DinFil implements Directions  {


       /*
        * Under her starter programmet ditt. Når vi starter
        * programmet vil datamaskinen lete etter ”main” og
        * starte alt derfra, så våre kommandoer må komme inni
        * der.
        */
      public static void main( String [] args )  {

         // Her kommer din kode:



      }

      // Gjør kartet synlig, sett inn navn på kartet under

      static  {


        World.readWorld( “mitt_kart.kwld” );

        World.setVisible( true );

       }

    }
```

Figur 22: Malen studentene skulle starte ut i fra.

En ser at dette programmet ikke inneholder noen kode knyttet til objektet. Dette skulle være et poeng. Første gang studentene skulle bruke malen skulle de erfare hvorledes de hentet dette programmet fra en fil og hvordan de skrev koden som opprettet et objekt. Samtidig kunne de registrere at objektet ikke foretok seg noe som helst. Det var først i første aktivitet de skulle gi instruksjon til at objektet skulle utføre noe.

7.3.4 Plan for innledning

Innledning til hver aktivitet skulle være basert på at kunnskap som var nødvendig i størst mulig grad skulle komme frem muntlig. Det vil si at det ikke skulle være behov for å lese skriftlig tekst for å løse problemet. Teksten skulle bare brukes som støtte i arbeidet.

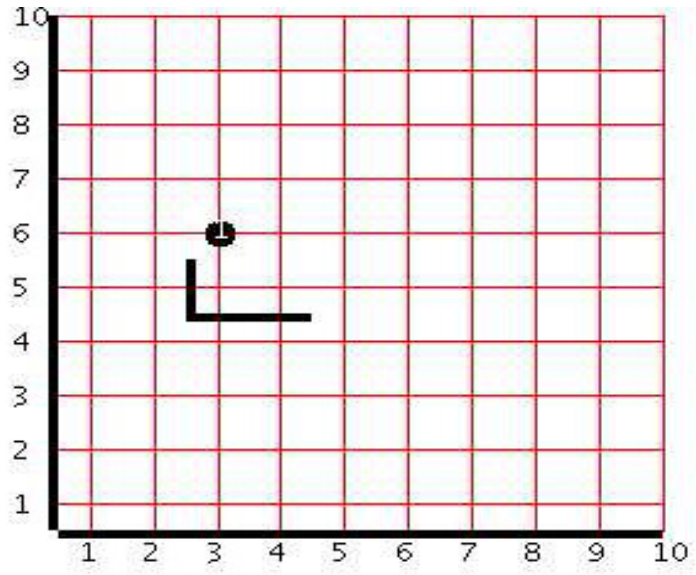
Aktivitet 1

Et objekt av typen ur_robot kan bestilles fra fabrikken ved hjelp av:
Ur_robot navn =new ur_robot (gate, aveny, retning, pipere);

Objektet forstår bare noen bestemte kommandoer som du kan fortelle den.

I denne aktiviteten skal vi lære hvordan vi kan fortelle objektet å komme forbi vegger og finne frem til piper. Objektet skal stå i krysset mellom 1. gate og 4. aveny.

En verden vi skal jobbe med her, ser ut som følger:



Figur 23: Eksempel på aktivitets beskrivelse.

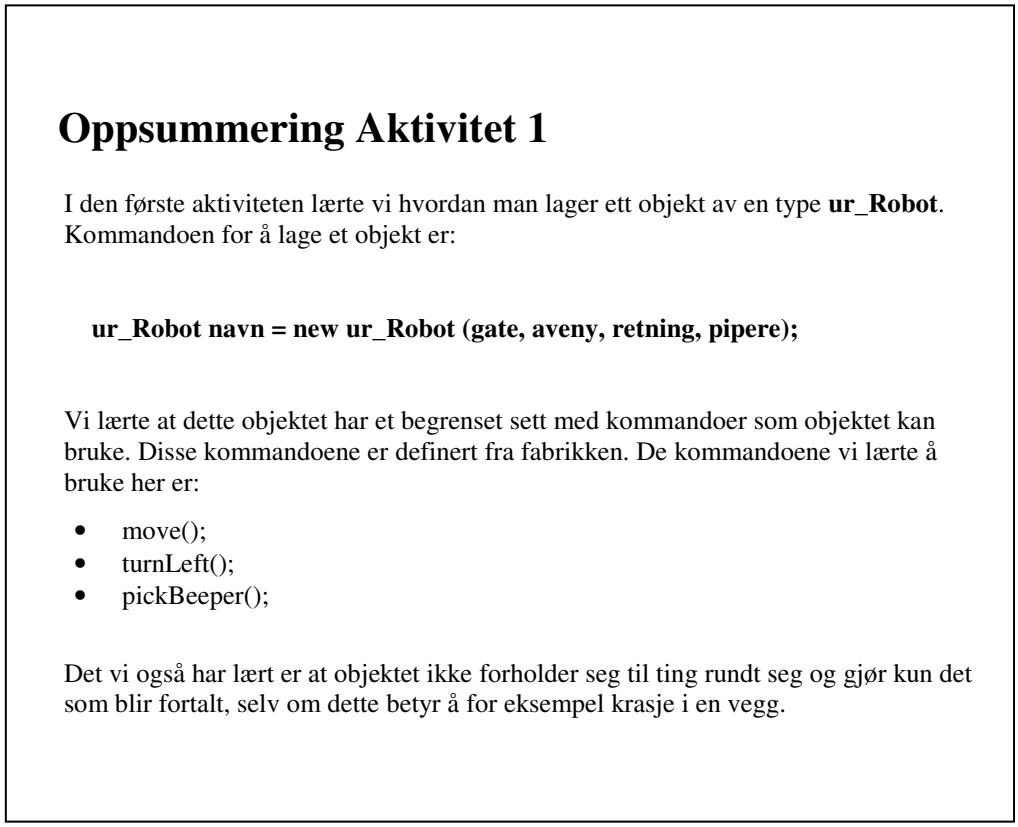
Ut i fra denne teksten kan det sees at her er det så lite informasjon at det for studentene kan være greit at det skulle være en muntlig kommunikasjon på forhånd. Denne kommunikasjonen skulle være interaktiv. Intensjonen var at alle som var til stede, både undervisningspersonalet og studentene skulle bidra. Målet var at utvekslingen skulle bringe frem mest mulig innsikt i den forestillingsverden som

deltagerne hadde. Med det menes her at undervisningspersonalet skulle få innsikt i studentenes forestilling om de begrepene som kunne komme til anvendelse i arbeidet, samt at studenten fikk en bedre innsikt i hvordan vi brukte begrepene. Der det var uklarheter skulle vi prøve å nå frem til noe som deltagerne kunne gi uttrykk for som å være omforent.

Det som var vanskelig var at vi bare skulle bruke interaksjonen i innledningen til å kommunisere om begrepene og problemet. I vårt opplegg var det vesentlig at studentene skulle prøve forslag og erfare hva som ble resultatet. Derfor måtte vi passe oss for ikke å gå for langt slik at innledningen fikk mer karakter av forelesning. Her tenker jeg spesielt på hensyn til temaer som det var meningen at studentene skulle erfare.

7.3.5 Plan for oppsummering

Oppsummeringen etter hver aktivitet skulle som tidligere beskrevet baseres på interaksjon med studenter, i tillegg skulle vi levere ut en kort enkel tekst.



Figur 24: Eksempel på oppsummering.

Her kan en se at teksten er meget kort, mer tenkt som stikkord. Dette var med hensikt, fordi vi ville bruke oppsummeringen for å bringe frem studentenes

erfaringer. Og ved at teksten er så pass kort håpet vi at studentene skulle føle et større behov for å delta i den muntlige oppsummeringen.

Oppsummeringen skulle være en åpen diskusjon hvor studentene kunne bringe frem sine egne antagelser om begrepet i øvelsen. Vi skulle bruke tavle for å tegne illustrasjoner og skrive på, dette siden det kan være lettere å samtale rundt en illustrasjon som alle kan se. Med svamp og kritt kan du endre på illustrasjonen ut i fra innspillene og vi kunne tegne endringsforslag som kan diskuteres. Dessuten følte vi at tavlebruken forsterker det uformelle. En setting som har en del uformelle elementer vil stimulere til en mer åpen dialog. Fra kurs vi hadde tatt i studiet hadde vi kunnskap om at det et pre med en åpen dialog for å stimulere til kritisk tenking, kreativitet og assosiasjon.

Vi skulle gjennom dialogen prøve å binde det nye begrepet til en sammenheng med de andre begrepene vi hadde jobbet med. Hensikten her var tosidig, vi skulle prøve å se hvorledes det nye begrepet henger sammen med andre begreper og vi ønsket å repetere tidligere tema. Repetisjonen skulle også brukes til etter hvert å bygge en klarere forståelse enn studentene på det aktuelle tidspunkt hadde.

8 Gjennomføring av eksperimentet

I dette kapitlet presenterer jeg gjennomføringen av eksperimentet. Dette gjøres ved å presentere hva studentene gjorde, mine observasjoner og jeg presenterer min analyse av disse, samt at jeg diskuterer dette i forhold til andre studier. Det finnes flere måter å strukturere en slik beskrivelse på. Jeg har valgt å skrive slik at det skal være greit å sammenholde det planlagte (7 Planlegging av eksperimentet) med det som ble gjennomført. At gjennomføringen er beskrevet kronologisk med hensyn på tid, hevder jeg er vesenlig for å gi leseren innblikk i forløpet av studentenes utvikling under kurset. Dette gjøres for klarere å se i hvilken grad mine to hovedfokus i oppgaven, veiledning og studentdrevet progresjon, bidro til læring av de objektorienterte begrepene. Drøftingen av resultatene kommer i kapittel 9 Drøfting og funn. Jeg presiserer at forskjell i struktur mellom kapittel 8 og 9, begrunner jeg med beskrivelsen ovenfor og at flere av episodene og kommentarene fra studentene drøftes under flere av de objektorienterte begrepene.

Til sommerkurset ble det sendt inn en invitasjon til 150 vilkårlig valgte personer som skulle starte på informatikkstudier ved Universitetet i Oslo. Det ble gjort klart i invitasjonen at vi ønsket deltagere uten erfaring i objektorientert programmering. Vi hadde med bakgrunn i antall tilgjengelig maskiner, og for å ha muligheten til å gi en tilstrekkelig veiledning, fastsatt en grense på 20 deltagere. Vi fikk 12 stykker som meldte seg på. Da kurset startet møtte bare seks personer, alle menn mellom 19 og 23 år. Bakgrunnen til de seks var at de hadde brukt datamaskin i forskjellige sammenhenger, men fem hadde ikke erfaring eller tatt kurs i programmering ved hjelp av 3 generasjons verktøy. En av deltagerne hadde noe erfaring fra å bruke BASIC.

Det å plassere deltakerne slik at de var to stykker ved hver maskin gikk greit. Vi hadde dessverre ikke noen kamera tilgjengelig før etter første dag. Det betyr at den første delen kun er dokumentert via notater og hukommelsen til de som var til stede.

Studentene ble orientert om hvilke roller vi fra teamet hadde. To stykker skulle observere uten å være aktive deltagere i undervisningsprosessen. To skulle stå for å gjennomføre undervisningen. Vi beskrev også hva vi ønsket å observere og registrere, og hvorledes data ville bli gjort tilgjengelig for de aktuelle forskerne gjennom nettet ved universitetet.

Vi registrerte ikke noen negative reaksjoner hos studentene på at vi skulle observere arbeidet de gjorde. Det var først da videokamera senere skulle tas i bruk at vi fikk en reaksjon. En av deltagerne ville ikke at vi filmet vedkommende. Det ble da en avklarende diskusjon om hvorledes opptaket skulle oppevares og om hvem som hadde tilgang til dataene. Spesielt ble det fokus på sikkerhet på hvem som hadde tilgjengelighet til dataene. Etter at vi hadde vært gjennom denne runden, gikk han med på at vi filmet bakfra slik at ansiktet ikke kom med på opptaket og han ikke kunne gjenkjennes. Nå viste det seg senere under kurset, at han ikke gjorde noe for

å hindre at han kom med på opptaket. Han snudde ansiktet mot kamera og var aktiv i diskusjoner.

8.1 Første dagen

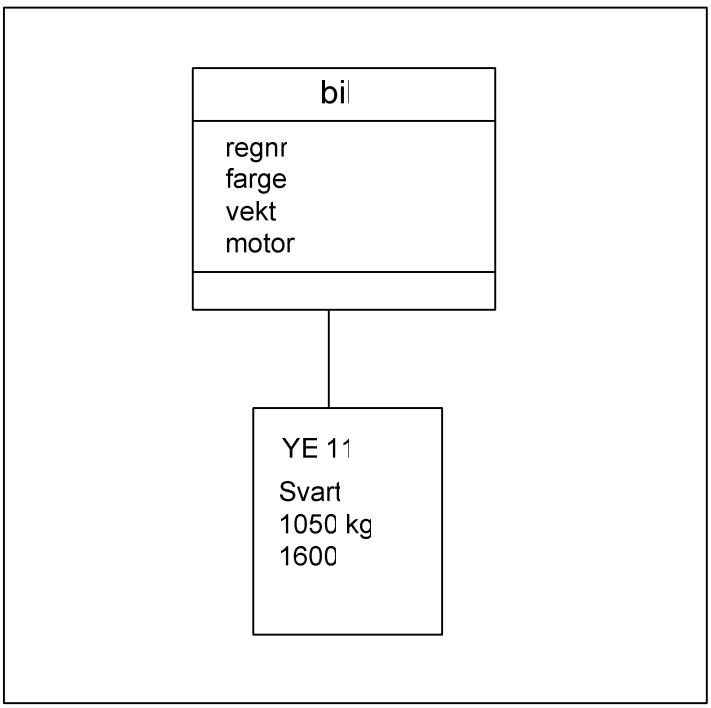
Vi holdt en innledning om kurset og hvilke interesser jeg og den andre kursholderen hadde, relatert til vårt hovedfagsarbeid. Vi fortalte litt om arbeidsformen vi kom til å anvende og vi beskrev hvilke forventninger vi hadde til målene vi hadde tenkt oss.

I tillegg til å lære deltagerne å programmere, ønsket vi å formidle hvorledes en datamaskin virket, uten å gå for detaljert til verks. Siden det senere i kurset kom spørsmål som hadde sin bakgrunn i en del av våre tidlige beskrivelser, tar jeg med litt om hva vi prøvde å formidle. Når selve kursfasen dro i gang, startet vi med å samtale en del rundt at datamaskiner ikke har noen intelligens og er ganske primitive. For å få maskinene til å virke må vi ha programmer som kommuniserer med maskinvaren. Som illustrasjon brukte vi Figur 20: Lagdeling med sirkel. Grunnprinsippene er basert på 0 eller 1 (dvs spenning /ikke spenning), legge sammen to tall, samt sammenligne to tall og finne hvilket tall som har størst verdi. Ved å bygge sammen mange instruksjoner kan vi få maskinen til å gjøre mange forskjellige jobber, så som styre et fly, en prosess i en fabrikk, bruke et regneark, kjøre et personalsystem, lønnsystem og så videre. Programmene blir utført ved at maskinen gjør instruksjoner en og en i sekvens. Maskinen sammenlignet vi med en vannkikkert, hvor du ligger i båten og ser ned i vannet. Du ser bare det som du ser i kikkerten, du kan ikke se hva som er neste instruksjon. Dersom du skal finne ut om det nærmer seg en båt må du se opp. Det ble også sagt at grunnen til at det kan se ut som om datamaskinen gjør ting i parallell, er at den er så rask at den rekker å gjøre flere oppgaver innenfor en meget kort tidsperiode. Denne fasen brukte vi for å skape kontakt og lede frem mot en aktiv dialog i kurset, dette gjorde vi ved hjelp av å bruke både rettede spørsmål og forespørsel om studentenes oppfattelse av begreper.

Når vi startet på beskrivelsen av Karel og hans verden ble det brukt en beskrivelse av en fabrikk som vi kunne bestille objekter fra. For å få studentene til å definere begrepet objekt brukte jeg spørsmålet: ”Hva er et objekt, Hva betyr ordet?” Med en slik innledning fikk jeg de til å prøve å tenke over ordet objekt, før de sa noe. De trengte litt tid, og jeg måtte følge opp med: ” Tenk på dagliglivet. Hva tenker dere på når dere hører ordet?” Dette følte jeg at jeg måtte gjøre fordi jeg fikk en fornemmelse av at setningen ”Hva betyr ordet?” gjorde at de kanskje trodde at vi var ute etter en formell beskrivelse av ordet. I ettertid kan jeg vel si at jeg kanskje skulle stoppet etter første setningen.

Bestillingen som det skulle bestilles objekter fra besto av det som vi beskrev som en arbeidstegning eller en mal. Med mal menes her sjablong, arbeidsmodell. Ovenfor studentene presenterte vi en assosiasjon med biler fra en bilfabrikk. Siden bestilling og produksjon av biler ble brukt i flere sammenhenger og vi også sammenlignet med andre tilsvarende prosesser, beskriver jeg eksemplet med bestilling av bil litt nærmere. En del bilmerker har modeller du kan bestille og du kan velge utstyr ut i fra en liste: Farge, interiørfarge, skinn /stoff i seter, radio, motor og så videre. Når

en bil er under produksjon finnes det en beskrivelse for det enkelte objekt. Denne beskrivelsen er laget ut i fra en mal. Her brukte vi en figur for å illustrere:



Figur 25: Klassebeskrivelse med et objekt

Det er verdt å merke seg at vi ikke tok med alt på figuren som vi fortalte kunne finnes på listen. Jeg gjør oppmerksom på at figuren vi brukte ikke er helt korrekt ide jeg bruker begrepet motor istedenfor ytelse og benevningen ikke er konsistent. Disse valgene ble gjort for å være illustrasjon og en del av en uformell kommunikasjon. Vi fortalte også at vi kunne legge inn egenskaper som utfører noe. Eksempler på dette er en funksjon som kontrollerer at bilen ikke spinner eller en hastighetssperre. Under denne orienteringen delte vi ut litt skrevet informasjon om Karel, hans verden (Vedlegg 1) og litt grunnlagsmateriale om hvorledes jobbe med Karel og hans verden ved hjelp av dataprogram (Vedlegg 1). Vi gav ikke studentene tid til å lese gjennom stoffet, siden vi via muntlig kommunikasjon, projektor eller tavle skulle ta opp de temaene de trengte i sitt arbeid.

Etter at studentene var etablert med et arbeidsbord startet vi den første øvelsen. Den kalte vi Aktivitet 1. Vi delte ut Karel 1 (Vedlegg 1) og Aktivitet 1 (Vedlegg 3). Oppgaven går ut på å plassere en robot og bevege den i en meget enkel verden.

Projektoren var et godt hjelpemiddel når vi skulle demonstrere hvordan de skulle hente malen (Vedlegg 2) og hvor de skulle putte inn den første linjen. Den første linjen opprettet objektet:

Ur_Robot karel = new ur_robot(1,4,North,0);

Som planen var, lot vi de kompilere og kjøre programmet for å se at objektet bare dukket opp på skjermen og ikke gjorde noe som helst. Her brukte vi muligheten til å

referere til lagdeling i programvare som tidligere omtalt i Figur 21: Illustrasjon på kommunikasjon mellom lag, og si at neste lag tok seg av å presentere objektet på skjermen i riktig posisjon.

Navnet Karel forklarte vi som noe som entydig identifiserer objektet, på samme måte som et bestemt fødselsnummer brukes på en bestemt person. Vi la her til at flere personer kan hete det samme og derfor egner ikke navn seg til å entydig identifisere personobjekter i et datasystem. Dette var en forklaring som studentene godtok direkte, kanskje det kom av at de tidligere har vært borti at en person kan identifiseres med fødselsnummer. Studentene tok i bruk de begrepene vi introduserte nokså direkte. Dette kan komme av at vi la opp til å prøve og få studentene til å akseptere en unøyaktig bruk av begreper og etter hvert i kurset prøve å tilnærme oss en mer korrekt bruk av begrepene.

Den neste linjen i programmet: *karel.move();* forklarte vi med at Karel bare kan bevege seg et skritt om gangen og at neste lag sørger for at figuren flytter seg et steg på skjermen. For å få Karel til å flytte seg mer enn et skritt, måtte vi gjenta kommandoen med tilsvarende antall linjer i programmet. Vi oppfordret studentene til å kompilere og kjøre programmet for hver gang de gjorde en endring i koden. Allerede på dette tidspunktet fikk de en forklaring om at dette ville hjelpe dem til å vite hvordan det siste de gjorde fungerte. Det vil si at de begrenset mulighetene for hvor det eventuelt kunne være feil.

Fra posisjon 1,4 (Figur 18: En enkel verden) med retning nord vil objektet kollidere med veggen. Noen spurte på dette tidspunktet om: ”Hva skjer når neste bevegelse treffer veggen?”. Vi ville at de skulle erfare dette og sa: ”Prøv”. Nå kunne vi bruke en forklaring om at objektet beveger seg i rommet uten å kunne registrere noe rundt seg. Her brukte jeg en demo med at jeg gikk et skritt og et skritt og kolliderte så med veggen. Studentene var klar over problemet og de fikk da instruksjonen:

turnLeft();

Etter å ha instruert roboten til å snu til venstre, lot de den bevege seg langs veggen og i posisjon 4,2 opplevde de at de måtte få roboten til å snu til høyre. Helt naturlig prøvde de *turnRight();* og fikk feilmelding. Da kunne vi bruke denne situasjonen til å påpeke og repetere at objektet bare hadde noen funksjonaliteter og *turnRight();* ikke finnes innebygd i laget under programmet de lager. Her ønsket vi at de etter hvert skulle oppdage at programmet deres ble langt, og senere skulle vi introdusere hvorledes de kunne forenkle programmet med å lage moduler.

Da oppgaven var løst hadde vi en oppsummering (Vedlegg 4) hvor vi ikke fokuserte på hvordan programmet virket, men på de begrepene vi hadde jobbet med: Opprette objekter, identifisere objekter, instruere objekter til å utføre noe, kompilere, kjøre program, prøve programmet for hver del de gjorde med endringer i koden. De hadde også opplevd å sende en beskjed til et annet lag, selv om dette ikke er en opplevelse som de kan se eller føle.

Vi delte ut ”Karel J, Neste skritt: Flere roboter” (Vedlegg 1) og ”Aktivitet 2” (Vedlegg 3), aktiviteten går ut på å plassere to roboter som skal beveges forbi et enkelt hinder for så å plukke opp to beepere hver (Vedlegg 3, Aktivitet 2). Mens vi

delte ut hadde vi en kort innledning. Her refererte vi til det vi hadde gjort med et objekt, identifisering og brukte figurene vi hadde på tavlen fra tidligere for å illustrere malen og objektbegrepene. Vi utvidet figuren med to objekter (Figur 17: Illustrasjon av klassebeskrivelse med objekter). Dette er ikke en standard måte å illustrere på, fordi den består av å kombinere to formelle konsept i UML diagram, nemlig klassediagram og objektdiagram. Vi klargjorde at vår måte å bruke figuren på, formelt ikke stemte med slik de i senere kurs ville anvende elementer fra figuren. I pausen senere på dagen spurte jeg deltagerne om de syntes det hadde fungert og om de hadde en sånn noenlunde oppfattelse av hva vi hadde snakket om. En representativ kommentar fra en student var: ”Det er enkelt å bruke mal for å lage objekter, når vi gjør det i programmet går det greit. Men du, hvordan bruker du tegningene når du skal utvikle datasystemer?” Dette påpekete to ting. For det første representerte den at tilbakemeldingene var klare på at oppfattelsen de fikk under innledningen min, stemte med det de opplevde mens de jobbet videre. For det andre var det en overraskelse for oss som holdt kurset at vår omtale av elementer som senere i utdanningen skulle brukes for å utvikle datasystemer, skulle vekke slik nysgjerrighet for å lære mer. Vi hadde regnet med at deltagerne ville ha mer enn nok med å fokusere på det vi gikk igjennom.

Med hjelp av å bruke vårt eksempel fra bilproduksjon og det å identifisere personer som jeg har beskrevet tidligere, hadde vi en samtale som skulle klargjøre at vi ut i fra samme mal kunne bestille flere individer som hver hadde sin egen identitet.

Det kom et spørsmål: ”Kan objektene bevege seg i parallell, det vil si på samme tid?” At spørsmålet kom var egentlig ikke så overraskende med bakgrunn i at vi hadde beskrevet datamaskinen som en maskin som gjorde instruksjoner i sekvens. Da måtte vi si at i utgangspunktet kunne datamaskinen ikke gjøre instruksjoner i parallell, men flere datamaskiner som jobber sammen kan utføre flere jobber samtidig. Når det kan se ut som en datamaskin gjør to eller flere oppgaver i parallell, så kommer det av at den er så rask at vi ikke legger merke til at den skifter mellom å gjøre biter på oppgavene. Her brukte jeg som eksempel tidsdelingen som er på flerbrukermaskiner. Jeg nevnte kort at den samme teknikken brukes når studentene senere i utdanningen lærer å programmere tråder. Nå hadde vi ikke flere maskiner i parallell, så derfor måtte vi holde oss til at instruksjonene ble utført i sekvens og at Karel J systemet hadde innebygd en forsinkelse mellom hver instruksjon som skulle utføres. En student foreslo da at vi måtte instruere maskinen om å flytte et objekt og så flytte det neste objektet og så videre.

Det var overraskende at studentene hadde så klar oppfattelse av oppgaven som de skulle jobbe med, og hva som var poenget i begrepsforståelsen. Vi hadde regnet med at vi måtte veilede mer på oppgaveproblemet og på begrepenes tolkning. Spesielt siden det utleverte materialet var svært kortfattet.

Studentene hadde litt problem i starten med å huske på at de måtte bruke riktig navn (identifikator) på det objektet som skulle utføre en instruksjon. Når det kom frem feilmeldinger eller at feil objekt beveget seg, kom de raskt med et korrigert forslag. På forespørsel fra meg til de enkelte gruppene, gav studentene uttrykk for at de fikk programmet til å utføre de bevegelsene på roboten som de ønsket. Mine observasjoner støtter opp om disse uttalelsene. Selvsagt ble det enkelte skrivefeil, men de rettet, kompilerte med riktig fil for verden og kjørte programmet uten at vi

registrerte noen uoverensstemmelse mellom utførte operasjoner og hva de uttrykte at de prøvde å få roboten til å gjøre.

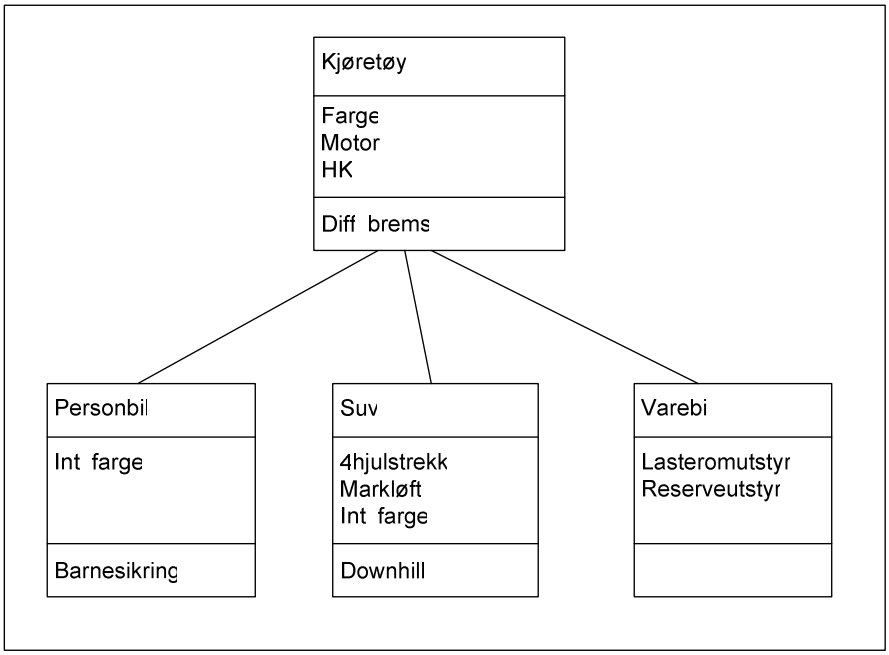
Oppsummeringen (Vedlegg 4) hadde fokus på identifikasjon av objekter og at objektene tilhørte samme mal. Vi bandt dette til det som vi hadde snakket om tidligere i forbindelse med objekter. Vi var kort innom at i starten på aktiviteten hadde de litt problem med å definere filnavnet for riktig verden.

Etter en lunsjpause fortsatte vi, og delte ut Karel J 3, Undertyper og metoder: Mer tilgjengelighet og oversikt (Vedlegg 1) og neste oppgave som var Aktivitet 3-1 (Vedlegg 3). I denne aktiviteten er sub-klasser det nye temaet. Dette innførte vi med å starte med begrepet spesialiserte objekter.

I introduksjonen tok vi opp med studentene at programmene ble lange, lite oversiktlig og vanskelig å jobbe med, dersom vi skulle programmere denne aktiviteten på samme måte som vi hadde gjort hittil. Som eksempel brukte vi at de tok i bruk *turnLeft();* instruksjonen når de ville at objektet skulle snu til høyre. Ved hjelp av den tidligere nevnte lagmodellen beskrev vi at programvaren i en datamaskin er laget med lag som kommuniserer med hverandre, og at jo høyere opp i lag vi kommer dess lettere er det å kommunisere. Dette fordi vi bygger på de tjenestene som de underliggende lag kan levere. Karel er et slikt underliggende lag som er ferdig laget fra leverandøren, og vi kan bare bygge nye tjenester basert på de som allerede finnes i Karel.

Her fikk vi et spørsmål om det gikk an å forandre i koden til Karel-systemet. Det vi svarte var at det kan være mulig, men at vi da måtte ha kildekoden til Karel, pluss god kunnskap om hvorledes Karel var bygget opp. Siden vi ikke hadde tid eller det som trengtes for å endre i kildekoden, skulle vi anvende det som fantes tilgjengelig når vi jobbet med øvelsene.

Vi hadde til nå unngått å anvende ordet klasse når vi hadde snakket om objekter, vi innførte nå ordet og brukte figuren Arv som illustrasjon for klasser og trakk sammenligninger med Figur 17: Illustrasjon av klassebeskrivelse med objekter.



Figur 26: Illustrasjon av arv

Vi ser her at notasjonen er delvis UML-notasjon. Dette for at jeg ikke ville at studentene skulle fokusere på notasjonen, men heller på relasjonen mellom boksene. Det kan se ut av figuren som metodene er utstyr som kan monteres, men vi beskrev metodene som programdeler som lå i bilens datamaskin og som det kunne åpnes for tilgang til.

Som eksempel på arv brukte vi bilfabrikanter som lager biler av forskjellige modeller, men hvor bilmodellene arver visse egenskaper fra klassen bil i fabrikken og i tillegg får noen egne egenskaper. Motorer brukes som regel i flere modeller mens karosseriet bare brukes på den aktuelle modellen. Vi brukte også som eksempel klassen person og arv til klassene dame og mann for å få et utvalg som studentene kunne assosiere til. Vi nevnte også noen andre varianter av klasser med arv under diskusjonen i innledningen. Når vi tok det problemet studentene skulle jobbe med, sa vi at vi ønsket å ha en ny type robot kalt ur_robot2. Denne skulle være en variant av ur_robot typen. Her beskrev vi at en ur_robot2 arver alle egenskaper som ur_robot har og i tillegg kan vi lage nye egenskaper. Vi ville at de skulle lage en ny egenskap *turnRight();* slik at den kunne snu til høyre.

Vi brukte en projektor for å vise hvorledes vi kunne programmere en robot variant ur_robot2.

Det ble en del diskusjon rundt konstruktøren, vi oppfattet det slik at studentene hadde vanskelig for å for akseptere konseptet. De syntes den var for komplisert. Vi sa de måtte akseptere utformingen og at vi ville komme tilbake til temaet senere, dette ble godtatt som en foreløpig beskrivelse.

Etter dette gikk øvelsen greit og oppsummeringen viste at de hadde en oppfattelse av hva en klasse var og hva et objekt var. Det var også en oppfattelse av at objektene hadde en identitet som vi kunne bruke for å instruere objektene

individuelt til å bevege seg etter en kodedel som objektene hadde felles. Nå viste det seg senere at studentene av og til brukte begrepene i feil sammenheng, men dette kunne vi kommentere ved å henvise til de diskusjonene og arbeidet som de hadde utført i denne øvelsen.

Aktivitet 3-2 (Vedlegg 3) hadde til hensikt at studentene skulle se nytten av gjenbruk av moduler, samt se at det kunne være nyttig å ha en plan for hvorledes programdelene ble laget.

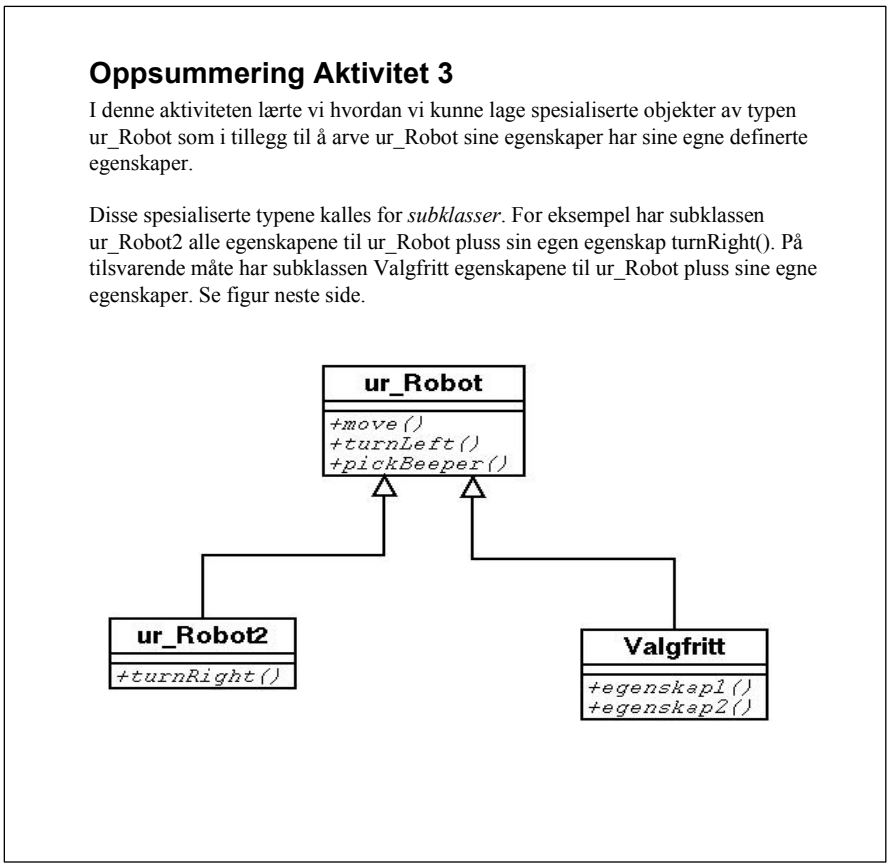
Under innledningen var det stor enighet om at 750 linjer for å få et objekt til å gå opp trappene i en blokk virket lite smart og hintet tok de fort. Vi ønsket at gruppene skulle jobbe med utfordringene hver for seg og stoppet innledningssamtalen før de kom så langt at de begynte å konkretisere problemstillingen.

Studentene fant raskt et mønster, men de valgte forskjellige måter å bruke mønsteret. En gruppe ønsket å telle trinnene og så skrive koden det antall ganger som trengtes for å løse oppgaven. Etter at de fikk spørsmål om ikke det ble mange linjer med kode, tok de i bruk erfaringen fra Aktivitet 3-1 (Vedlegg 3) der de laget en metode for å utføre *turnRight();*. De foreslo at de kunne lage en metode for et trinn og så gjenta dette det nødvendige antall ganger. Når de jobbet med oppgaven opplevde de at roboten etter å ha plukket opp første element fortsatte rett frem inn i veggen. Det tok litt tid før de hadde klart for seg at objektet startet med retning nord og etter første pickup sto det med retning vest, slik at neste kall startet med et annet utgangspunkt enn første kall. Når de fikk dette klart for seg gikk det greit å lage metoden slik at utgangspunktet var likt ved hvert kall til metoden. En annen gruppe brukte ur_Robot2 klassen fra forrige aktivitet. Gruppen bygget videre på *turnRight();* metoden, men de ble i tvil om de bare kunne kalle metoder fra klassen eller om en metode kunne kalle en metode som lå i en annen klasse. Etter en liten diskusjon med gruppen, hvor vi viste til at det ville være vanskelig og lite oversiktlig å lage nye klasser for alle nye funksjonaliteter vi skulle ha, tok vi en felles diskusjon på det temaet ut i fra at vi regnet med at flere kunne ha problem med avklaringen av dette. Etter dette gikk oppgaven greit for gruppen.

En tredje gruppe brukte Karel J 3, Undertyper og metoder: Mer tilgjengelighet og oversikt (Vedlegg 1) dokumentet som en mal og når de fikk feil klarte de relativt greit å rette og komme frem til en løsning som fungerte.

Denne oppgaven mente vi var sentral når det gjaldt å vise objektorientert tankegang, så under veiledningen brukte vi en del tid på å diskutere forskjellige aspekter og innspill som studentene kom med.

Under oppsummeringen ønsket vi å få frem at det er flere løsninger og at det som regel er et valg mellom alternativer og at det sjelden er slik at en løsningstrategi er den eneste riktige. Derfor lot vi gruppene fortelle kort om hvorledes de hadde løst oppgaven. Det ble litt diskusjon rundt løsningene, men vi mangler som sagt videoopptak som kan brukes for en mer nøyaktig analyse av dette. Oppsummeringen tok også opp bruk av egenskaper i objekter, spesialiserte egenskaper og hvorledes disse hang sammen med begrepet arv. Som illustrasjon delte vi ut:



Figur 27: Oppsummering Aktivitet 3

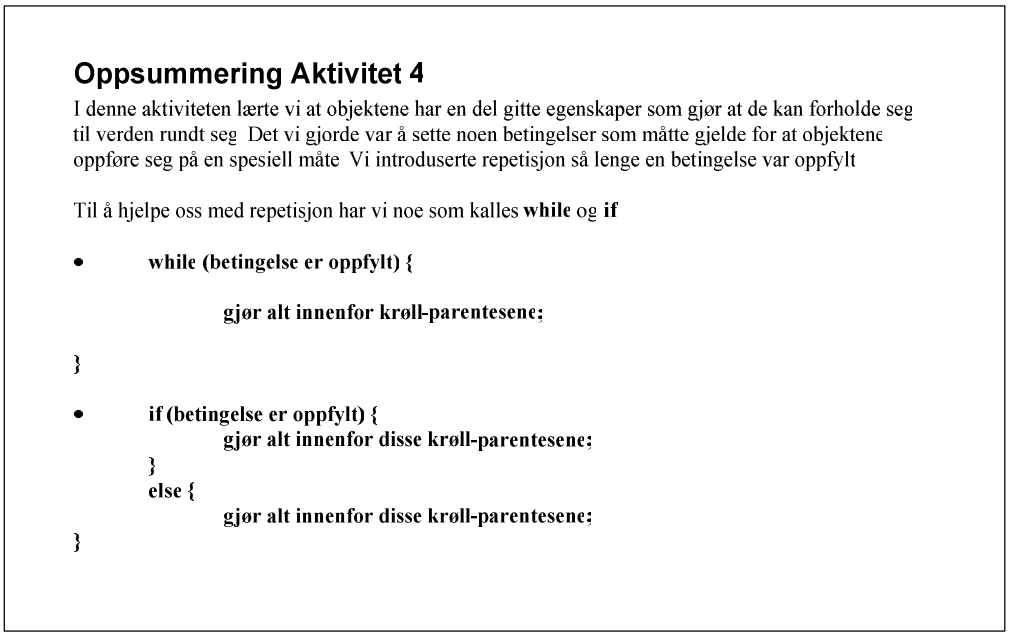
Legg merke til at figuren her er mer korrekt etter UML standard enn Figur 26: Illustrasjon av arv. Dette ble ikke kommentert av oss, bare tatt i bruk.

Vi delte ut Karel J 4 Mer teknisk (Vedlegg 1) og Aktivitet 4 (Vedlegg 3). I denne aktiviteten skulle de fortsette opp en trapp og stoppe ved å teste på en betingelse før objektet traff taket.

Under den innledende samtalen foreslo noen at de skulle gjøre det samme som i Aktivitet 3-2 (Vedlegg 3), bare det at de skulle bruke en teller som telte antall trinn. Da påpekte vi at dersom de skulle bruke det samme programmet på en annen trapp med et annet antall trinn, måtte de endre i programmet. Vi påpekte at de kanskje skulle se etter om det var andre ting de kunne bruke som betingelse. Da noen påpekte at taket kunne brukes, gikk vi såpass langt at vi sa at når de hadde gjentagelser skulle de lete etter noe som var et brudd på gjentagelsene, og så bruke det som betingelse for å teste. Siden oppgaven hadde et nytt moment, nemlig test for løkker og studentene lett kunne oppfatte at vi hadde flere alternativer, brukte vi pseudokode på tavle mens vi diskuterte og prøvde å illustrere konsekvensene av alternativene som kom frem. Da vi hadde kommet frem til at en while løkke ville egne seg, lot vi gruppene jobbe selvstendig med aktiviteten. Selv om de jobbet selvstendig fulgte vi med og stilte av og til enkelte av de tre spørsmålene bare for å få de til å være bevisst på hvorfor de valgte den koden de brukte. Personlig synes jeg det var helt greit når en av deltagerne sa: ”Vi har vel ikke noen spesiell grunn, vi vil bare prøve hva som skjer dersom vi prøver med …..”. Dette oppfatter jeg som en

ærlig uttalelse istedenfor å prøve å konstruere en eller annen forklaring. Noe som lett kunne skje siden vi hadde vært så klar på at det var viktig å vite hva en ønsket å oppnå med en kodebit som en skulle bruke.

Det var noen som under arbeidet med aktiviteten lurte på else og hadde litt problem med å skille klart mellom forskjellen på while og if – else setninger. En lurte på om de kunne bruke else i en while setning. Dette tok vi opp som et fellestema, samtidig som vi sa fra om at vi ville komme tilbake til dette i oppsummeringen. Resten av arbeidet med aktiviteten gikk meget greit. De fant frem til aktuelle begreper som kunne bidra i løsningen og de hadde gode begrunnelser for sine valg. Når de valgte feil, oppdaget de fort at de hadde valgt feil og kom da raskt opp med et bedre forslag til løsning.



Figur 28: Oppsummering Aktivitet 4

Under oppsummeringen hadde vi først en meget enkel gjennomgang av prinsippene i de to setningstypene, før vi hadde en mer åpen diskusjon om muligheter og hvorledes de hadde løst oppgaven. Så tok vi en enkel repetisjon av hovedprinsippene vi hadde behandlet denne dagen. Her fokuserte vi på de enkle forklaringene og begrepene vi hadde brukt samtidig som vi prøvde å få frem relasjonene mellom begrepene.

Etter første dag, hadde vi to som holdt kurset og de to observatørene en kort oppsummering av kursets første dag. Konklusjonen i denne var at vi hadde fått en raskere progresjon hos studentene enn vi hadde regnet med og skulle forsette opplegget med en endring. Endringen vi skulle ha til neste dag var en ekstra oppgave som skulle bygge på Aktivitet 4 (Vedlegg 3).

Vi var nokså enig om at verktøyet KarelJ var meget godt egnet for den type opplegg som vi brukte, idet det var lettere å snakke om begrepene og knytte disse til hva de så på skjermen av verden og robotens bevegelse. Det at det var lettere å snakke om

begrepene, gjaldt både diskusjonen internt i parene og i diskusjonen med instruktørene. Egnetheten var også basert på at studentene hadde en mal som de skulle fylle inn i, og siden KarelJ gir lite kode i den type oppgaver som vi brukte, forsterker dette mulighetene for å fokusere på det vi la opp til som vesentlig, nemlig studentenes begrepsforståelse. Vi registrerte også at studentene brukte objektorienterte begreper i sine samtaler og vi hadde fått de til å fokusere mer på begrepene enn på syntaks i koden.

8.2 Andre dagen

På den andre dagen hadde vi fått montert videokamera for å gjøre opptak av to av gruppene. Vi startet dagen med en repetisjon av stoffet fra første dag. Denne tok rundt regnet 25 minutter. Forskjellen fra første dag, var i det vesentligste at mens vi i dag en fokuserte på begrepene og sammenhengen mellom dem, prøvde vi under dag to å assosiere til andre erfaringer, samt bygge på varianter av det vi hadde brukt første dagen.

Oppgaven vi skulle starte med ble kalt Aktivitet 5 (Vedlegg 3) og var basert på Aktivitet 4 (Vedlegg 3). Hensikten var å tilføre en annen type aktivitet enn de øvrige oppgaver, som alle var basert på å observere en robots bevegelser. Denne aktiviteten skulle være en litt mer tradisjonell type oppgave og teksten var:

Bruk samme kart som i aktivitet 4. Bruk den utleverte klassen, plukk opp pipere og skriv ut på skjermen hvor mange pipere Karel har plukket opp.

I Karel's verden er en piper et tegn som vises på skjermen som en sort skive med et tall (Figur 29: Aktivitet 4.2). Siden vi syntes studentene hadde klart seg så bra første dagen, ville vi prøve hvordan de klarte seg med et problem som var annerledes utformet, i den betydning at det var en mer tradisjonell programmeringsoppgave. Siden oppgaven stort sett var basert på elementer som var brukt før, delte vi bare ut oppgaven og lot de jobbe med den. Vår veiledning gikk ut på at vi etter en tid gikk rundt og så på hva de jobbet med. Der vi følte behov for det, brukte vi de tre spørsmålene (6.3 3 spørsmål og 7.2.5 Veiledning) og bare dersom vi følte at det kunne bringe studentene videre. Selvsagt hadde studentene alle muligheter til å spørre om det de følte behov for, men vi registerte på dette tidspunkt aksept for at vi brukte våre tre spørsmål for å lede de videre. Det var bare ved enkelte tilfeller når studentene hadde problem med å skrive syntaks at vi følte behov for å fravike de tre spørsmålene.

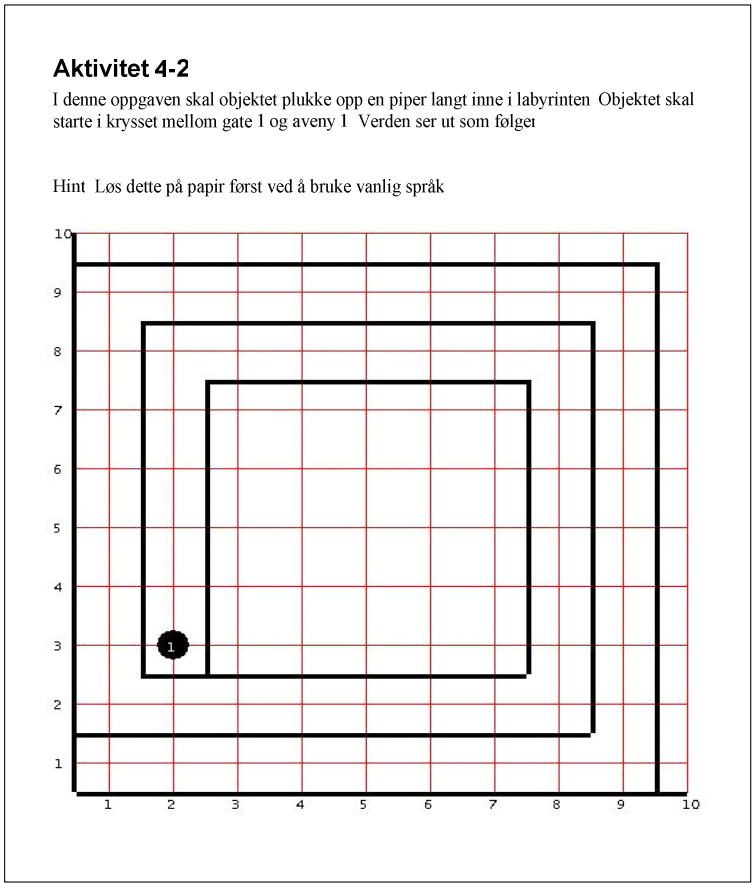
Gruppene kom godt i gang med å foreslå løsninger, men slet med å huske bruken av begrepene på rett måte i forhold til hva de ønsket å oppnå. Dette kom tydeligst til uttrykk ved at de brukte while og if feil. Et eksempel på dette er at en gruppe kom frem til at de skulle ha en setning med betingelse og valgte da å skrive while. De slet med å få while setningen til å fungere og tok derfor kontakt for å få veiledning. Det de da verbalt beskrev, var tydelig på at de skulle bruke betingelsen til å velge om en setning skulle utføres, det vil si en if setning. Veiledningen gikk da ut på å referere til begrepene vi hadde jobbet med dagen før og beskrivelsene vi hadde diskutert. Her fikk jeg studentene til å beskrive hvorledes vi hadde brukt begrepene.

Dette var nok til at de valgte riktig begrep og brukte det på riktig måte i forhold til hva de hadde til hensikt å utføre.

For å utvikle mer innsikt i hvorledes anvende moduler fra et program, over i lignende situasjoner i andre program, endret vi litt på kartet med å legge piperne ut på kartet i noen varianter. Et eksempel på dette er: Horisontalt mot høyre fra den øverste piperen i kartet til Aktivitet 4 (Vedlegg 3).

Mye av tiden til gruppene gikk i starten av aktiviteten med til å repetere kunnskap om begrepene, og deres anvendelse i programmering. Dette var litt overraskende, i det de dagen før hadde vist så god oversikt over begrepsforståelse, og hvorledes begrepene skulle anvendes i aktivitetene. Vi brukte derfor tid på å klargjøre begrepene gruppevis, og enkelte av begrepene tok vi i plenum og hadde en felles diskusjon rundt dem. Nå viste det seg at de etter at de hadde fått et begrep klart for seg, var det ikke så lett å bruke det riktig neste gang det ville vært naturlig for en erfaren programmerer å bruke det. Men dess lenger ut i oppgaven de kom, dess bedre ble de på det å gjenkjenne situasjonen og anvende begrepet riktig.

Siden Aktivitet 5 (Vedlegg 3) var laget kvelden før, ble benevningene vi brukte på aktivitetene i litt gal rekkefølge. Vi delte nå ut Aktivitet 4-2 (Vedlegg 3).



Figur 29: Aktivitet 4-2

Vi hadde ingen muntlig presentasjon når vi delte ut aktiviteten. Vi syntes studentene skulle prøve seg på en oppgavetekst som ikke skulle brukes for å lære nye begreper.

De skulle bare anvende den kunnskapen de allerede hadde jobbet med tidligere i kurset.

Denne oppgaven mente vi kunne se enklere ut for studentene enn vi mente at den var. Derfor ba vi om at de først løste oppgaven på papir med pseudokode. Dette for å forsterke opplevelsen at det å analysere et problem og så lage en grov skisse av et løsningsforslag, kunne hjelpe dem til å unngå noen av problemene som kan komme dersom de startet direkte med å kode.

Oppgaven har utfordringer i at vi ikke kan vite når roboten kan treffe på en piper og vi vet heller ikke hvor mange ganger den må vende seg til venstre. Dette fordi vi ikke vet hvor langt inn slutten på den spirallignende labyrinten er.

Studentene startet med å prøve på papir og dette arbeidet tok utgangspunkt i det konkrete kartet som de skulle bruke. Dette fordi de regnet med at de skulle skrive et program som skulle bevege roboten slik kartet viste, det vil si med det antall skritt og venstre svinger som de så på kartet. I veiledningen kunne vi henvise til at oppgaveteksten sa: ”Langt inne i labyrinten” og at det betydde at de måtte se på oppgaven som om roboten ikke hadde mulighet for å se i fugleperspektiv. Vi hadde jo før snakket om at roboten bare kunne registrere om en vegg var rett foran, eller den sto oppe på en piper. Dette skulle lede studentene til å få en klarere fornemmelse av at de måtte lage en mer generell løsning. Vi benyttet muligheten til å henvise til det vi tidligere hadde sagt om å dele inn et problem i mindre delproblemer, for så å jobbe med disse. Vi måtte gi de litt veiledning i hva vi mente med vanlig språk. Vi ville få de til å skrive ned løsningen i grove trekk i et språk som de selv forsto. Dette fungerte bra, i det de skrev og diskuterte prinsippene på en måte som jeg hevder brakte dem nærmere en løsning. Dette på tross av at det de skrev var langt fra det en dreven programmerer vil kalle pseudokode. Det kommer mer om dette senere når jeg ser litt på forskjellen i arbeidsform hos gruppene. Dessverre har vi ikke noen dokumentasjon på det som ble gjort på papir. Det ville ha vært interessant å se nærmere på hvorledes studentene jobbet med å skisse opp et løsningsforslag på papir og sammenligne dette med den endelige løsningen de laget i kode på maskinen.

Etter dette innledende arbeidet gikk arbeidet med aktiviteten greit. De gav i oppsummeringen tilbakemelding på at de var fornøyd med at vi brukte våre tre spørsmål og prøvde å få dem til å tenke frem forslag og forbedringer selv. Eksempel på dette var da en gruppe laget en kode som flyttet roboten 8 skritt til høyre, snudde den til venstre, flyttet den 8 skritt oppover og min kommentar var: ”Hvorfor gjør dere dette akkurat sånn? Hvordan passer dette inn resten av programmet?” De begynte å forklare hvordan de tenkte løsningen og oppdaget selv at roboten bare skulle gå åtte skritt rett frem tre ganger før antall skritt ble redusert. En foreslo å bruke en løkke som ble redusert med en for hver gjennomgang. Så sa en: ”Vi kan bruke den testen om den har veggen foran seg, og så snu til venstre.” De prøvde å implementere dette, da gruppen ropte på meg. ”Skal vi regne med at roboten vet at den skal svinge til venstre eller vet den ingen ting?” I dette ligger det at de referer til Aktivitet 5 (Vedlegg 3) hvor de skulle forestille seg at roboten ikke kan se kartet i fugleperspektiv. Gruppen endte opp med å først lage en versjon hvor roboten alltid svingte til venstre, for så å forbedre programmet med å sjekke til høyre, rett frem og til sist til venstre.

I oppsummeringen før spisepausen tok vi en gjennomgang av det å dele opp et problem i flere mindre problem og det å beskrive delene med hjelp av pseudokode. Vi uttrykte at denne pseudokoden kunne brukes til lettere å få oversikt over forslag som vi hadde utarbeidet. Vi snakket også om å generalisere kode fremfor å ha løsninger som bare kunne løse et spesifikt problem. Vi repeterte også forskjellen på valg (if) og gjør inntil (while).

Etter pausen hadde vi en oppsummering av de begrepene vi hadde brukt så langt, Vi tok med en kort definisjon for hvert begrep og vi hadde en kort diskusjon om problemene vi hadde behandlet i aktivitetene.

Vi hadde planlagt å la studentene prøve å programmere i en totalt annerledes setting ved å bruke Robocode (7.3.1 Valg av verktøy). Dette er et system som er betydelig mer komplekst i sin oppbygging og har en høyere inngangsterskel. Med dette mener jeg at det krever at brukeren for å komme i gang, må skrive betydelig mer og komplisert programkode enn i systemet til Karel J (7.3.1 Valg av verktøy). Vi var litt spent på hvorledes studentene ville takle at vi byttet system å jobbe i, selv om vi la opp til at det var de samme begrepene som skulle anvendes. Selv om vi regnet med noen problemer i starten, så regnet vi med at det skulle gå greit for studentene siden de hadde vist så stor fremgang i å anvende begrepene.

Vi startet med å fortelle litt om hva som var prinsippet i spillet og hvorledes de skulle lage en tank og knytte egenskaper til denne. Dette gjorde vi med å henvise til at det ofte kan være lurt å benytte gjenbruk. Vi fikk derfor studentene til å se på en av de ferdige definerte robotene i systemet. Vi gikk såpass langt at vi hintet på en robot som hadde en rimelig enkel programkode. Dette gjorde vi fordi noen av de ferdige robotene, har mange metoder som det kan være vanskelig å sette seg inn i. Studentene var meget engasjert og startet med entusiasme på aktiviteten. De skulle her lage en robot som skulle delta i krigen og prøve å ødelegge fiendens tank.

Vi oppdaget fort at dette var en betydelig vanskeligere oppgave. Det å sette seg inn i kode som andre hadde laget og så anvende denne på egen tank, ble et større problem enn vi hadde regnet med. Se gruppenes kode fra aktiviteten (Vedlegg 5) . Vi hintet på at det kunne være lurt å se på hva en tank gjorde og så lete i koden for å finne de delene de ønsket å bruke. Her var det slik at tanken beveget seg raskt og gjorde mange forskjellige bevegelser før spillet var over. Dette og at det ikke var mulig å få justert hastigheten på bevegelsene eller å få til en skritt for skritt kjøring av programmet, gjorde at det for studentene ble vanskelig å oppfatte hva som egentlig skjedde under spillet. Når de så skulle se på koden ble det for vanskelig å assosiere en bevegelse til det å finne kodebiten som stod for utførelsen.

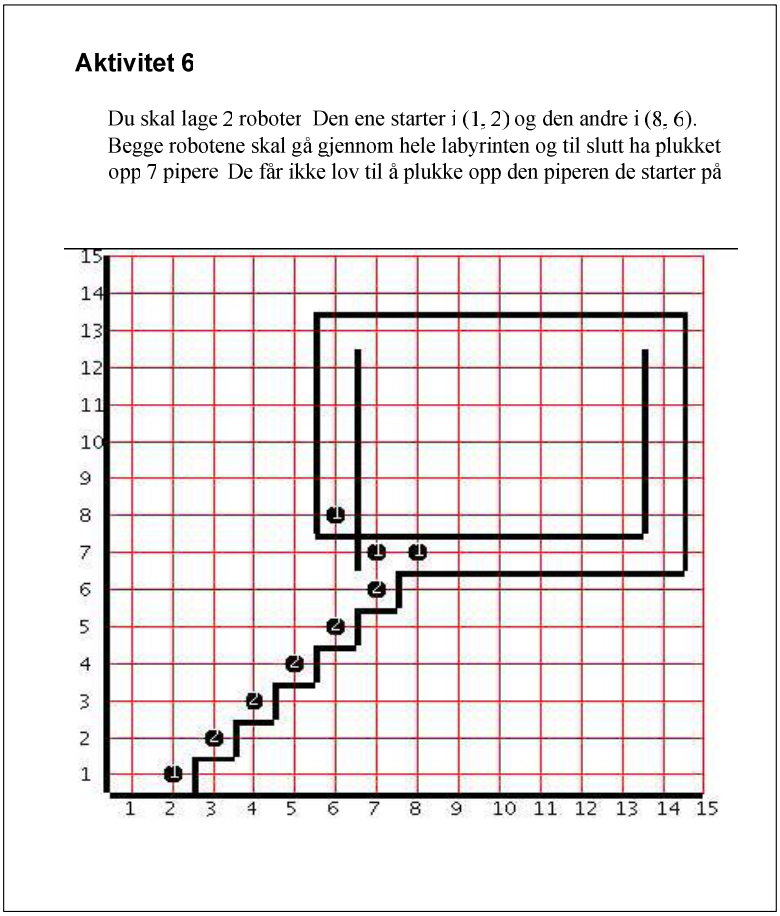
Gruppene fikk laget en enkel tank og lagt inn noen få metoder, men slet med å finne passende metoder utover dette. Vi fant at spranget i kompleksitet var så stort at studentene hadde vanskelig for å anvende begrepene. Slik jeg opplevde det var dette ikke fordi de hadde vag begrepsoppfattelse, men mer fordi kompleksiteten i programmeringen ble for vanskelig å håndtere med den erfaringen de hadde med programmering og innefor de tidsrammer vi hadde disponible.

Under oppsummeringen sa studentene at det var vanskelig å finne frem i koden som de skulle bruke som mal. For det første beveget roboten som var mal seg så raskt, at det var vanskelig å kunne assosiere til koden for malen. For det andre gav de uttrykk for at koden var mye vanskeligere å forstå enn de var vant med fra tidligere i kurset. De enkle bevegelsene hadde gått greit å få til, men de litt mer avanserte hadde de slitt med. Deltakerne syntes det var morsomt å jobbe med Robocode og at det var tydelig at programmering kunne bli mye vanskeligere enn de trodde før de startet på aktiviteten.

Vi som holdt kurset og observatørene hadde en kort oppsummering da deltakerne hadde forlatt lokalet. I denne var ble det uttrykt enighet om at Robocode aktiviteten vår var for ambisiøs. Dette fordi den aktivitet som vi hadde lagt opp til, krevde mer erfaringsgrunnlag enn studentene hadde etter de foregående aktiviteter. Det mest positive med aktiviteten var at det gav studentene en erfaring med litt lenger programkode. Settingen med at det underliggende laget sine tjenester, ikke er like rikholdig og lett synbar var også nytt. I dette ligger det at studentene måtte programmere mer kode for at roboten skulle gjøre enkle bevegelser på skjermen.

8.3 Tredje dagen

Den tredje dagen kom i uken etter og bestod av to deler. Først en repetisjon hvor vi gikk gjennom begreper og hvorledes vi hadde anvendt disse. Her ble det vektlagt at formen skulle være en diskusjon som inviterte til at studentene deltok aktivt og kunne bruke denne anledningen til å avklare begreper som de var usikre på eller hadde forskjellige oppfatninger av. Måten diskusjonen ble drevet på var i hovedsak at det ble stilt spørsmål, så prøvde deltakerne å gi sine synspunkter til disse. Vår rolle var å hjelpe til med å avklare, utdype, og stimulere til assosiasjon rundt diskusjonen. Ved å stimulere til assosiasjon skulle vi hjelpe studentene å anvende eller kjenne igjen kunnskap de hadde. Eksempler på det er: Kjenne igjen muligheter for bruk av arv og gjenbruk av kode. Den neste delen var Aktivitet 6 (Vedlegg 3):



Figur 30: Aktivitet 6

Denne ble delt ut uten noen felles diskusjon. Gruppene klarte å løse oppgaven, men valgte forskjellig arbeidsform. Den ene gruppen planla hva de ville prøve, og så prøvde de via assosiasjon og leting å finne biter av kode som kunne brukes. Når de skulle teste en ny bit kode var den som regel med et slikt antall linjer at de klarte å holde oversikten og nokså raskt fant eventuelle feil og kunne korrigere disse (Vedlegg 6). Denne gruppen fikk problemer med gjenbruk av kode, ikke ved at koden gjorde noe annet enn de forventet, men at navngiving av filer og klasser ikke ble konsistent i det nye produktet. Dette hevder jeg ikke kommer av at de hadde problem med begrepene og anvendelsen, men for lite erfaring i å huske og sjekke konsistensen av anvendelsen av for eksempel fil og klassenavn. Min begrunnelse for dette bygger i det vesentlige på observasjonen av at denne type feil skjedde flere ganger for denne gruppen. Etter hvert oppfattet de nokså raskt hva feilen var og fikk korrigert den.

Den andre gruppen jobbet betydelig mer ustrukturert. De anvendte gjenbruk av kode, men i det nye produktet la de til og skrev mye kode mellom hver gang de kompilerte og testet. Dette gjorde at de brukte lang tid på å korrigere koden (Vedlegg 6). Når de fikk et problem, prøvde de noe annet. Et eksempel på dette er når de hadde en test og roboten beveget seg annerledes enn de håpet på, så ble kommentaren: ”Vi prøver med en while setning”. Nå hjalp ikke det og jeg fikk de til å beskrive for meg hva de ville at setningen skulle utføre. Under diskusjonen

skrev jeg opp syntaksen til if setningen og de foreslo selv at det var testbetingelsen som var problemet.

Denne gruppens arbeidsform kan være en indikasjon på at vi burde ha vektlagt modularisering og metoder for effektiv utvikling av kode i større grad enn vi gjorde. Kan det være at vår veiledningsform ikke fikk frem dette aspektet klart nok, eller er det andre metoder som kan bidra til å gi en mer effektiv læring av dette? Det kan på en annen side være oppgavenes form som ikke ga nok stimuli til å fremme dette.

Mine notater og videoopptakene fra tredje dagen viser mye konsistens i anvendelsen av de begreper som jeg ser på i denne oppgaven. Der det kan registres at de bruker begrepene feil, skjer det korreksjon internt i gruppen og de får raskt en felles forståelse.

8.4 Oppsummering etter kurset

Vi holdt en oppsummering på slutten av kurset. Oppsummeringen ble holdt i plenum i en åpen form slik at studenter, observatører og instruktører kunne stille spørsmål og komme med synspunkter. Den siste aktiviteten Aktivitet 4-2 gav de uttrykk for tok lang tid, og inneholdt mer enn de trodde i starten. De syntes den var lærerik og gav mindre frustrasjon enn Robocode aktiviteten, i den forstand at de fikk en god følelse av at de følte at de lykkes med resultatet. Dette er et klart eksempel på mestringsfølelse slik det er beskrevet i kapittel 4.2 Mestre.

Det ble klart uttrykt at de mente at de hadde en klar forståelse av de begrepene vi hadde jobbet med og dette stemte med de kontrollspørsmål vi stilte. Det at vi brukte de tre spørsmålene, ble beskrevet som positivt fordi: ”Vi måtte tenke”. Det ble også uttrykt at noen ganger forstod de ikke hva vi prøvde å lede dem til. Men som en sa: ”Når vi bare fikk prata litt, så lysnet det”.

Det at vi baserte oss på en konstruktivistisk læringsteori hvor studentene skulle konstruere sin egen kunnskap ble omtalt som meget bra. De var positive til at de ikke var avhengig av å lese så mye for å kunne starte på en aktivitet. Noe av det som ble omtalt som mest positivt var innledning og oppsummering av aktivitetene i plenum. Det ble også uttrykt at vi kanskje skulle ha hatt litt mer øving før et nytt tema ble introdusert. Dette gjaldt særlig Robocode aktiviteten. Når de jobbet med den hadde de oppdaget at programmering kunne være mer komplisert enn de hadde erfart til da.

Progresjonen i kurset ble beskrevet som meget bra i det de følte en passe blanding av utfordring og det å vite at hver aktivitet bare skulle bidra med et nytt problem. Dette siste gjorde at ” vi viste at vi kunne løse det, det tok av og til bare litt tid før vi oppdaget hva vi skulle bruke”. Det eneste unntaket var det som er beskrevet i forrige avsnitt om Robocode.

For oss instruktører opplevde jeg for egen del at det mest positive var at vi ikke hadde skremt noen av deltakerne som var med siste dagen, i det disse så frem til å starte på de ordinære kursene.

8.5 Hvordan fungerte gruppene

Vi ønsket at gruppene skulle jobbe mest mulig selvstendig og motta veiledning etter forespørsel fra studentene, samtidig skulle vi gå rundt og ha mulighet for å kommentere og spørre mens de jobbet med aktivitetene. Jeg kan se av observasjonene at når det gjaldt samarbeidet i gruppen, begrepsbruken i muntlig kommunikasjon og skriving av de enkelte setninger i koden, var det liten forskjell på gruppene. Men selv om vi la opp til en ganske tett oppfølging av arbeidet, viste det seg at de to gruppene som vi fulgte opp med videoopptak, tidlig viste en forskjell i formen de jobbet på. Dette er tidligere omtalt i kapitlet og dreier seg om programstruktur og størrelsen av programbiter som ble testet. Det henvises til programeksempler fra Aktivitet 6 (Vedlegg 6).

9 Drøfting og funn

Som jeg har skrevet i hypotesen min var mitt mål i eksperimentet, å prøve ut om en didaktikk som baseres på veiledning og studentdrevet progresjon, kan bidra til begrepsbygging av objektorienterte begreper (1.2 Problemstilling). Jeg prøver å bruke dette perspektivet når jeg drøfter mine funn. Kapitlet er delt i fire hovedpunkt. Rekkefølgen av punktene er valgt ut i fra at leseren først skal kunne få presentert i hvilken grad studentene lærte de objektorienterte begrepene. Så drøfter jeg i hvilken grad didaktikken som ble brukt kan ha bidratt til læring. Med bakgrunn i disse to hovedpunktene er det enklere å presentere mitt mål fra hypotesen, de to temaene: Veiledning og studentdrevet progresjon. Til slutt kommer det fjerde hovedpunktet hvor jeg presenterer noen erfaringer som jeg sitter igjen med etter eksperimentet.

9.1 Drøfting av læring av OOP begreper

Dette kapitlet er en drøfting med veiledningsperspektiv på hvorledes studentene lærte de objektorienterte begrepene som ble valgt. Først tar jeg for meg valg av begreper, før jeg ser litt på alternativene som ikke ble valgt, samt at jeg tar noen generelle betraktninger. Til slutt får de valgte begrepene hvert sitt kapittel.

De objektorienterte begrepene vi hadde fokus på var objekt, klasse, sub-klasser, modularisering og gjenbruk. Disse begrepene ble i hovedsak valgt av meg via en diskusjon med den andre som skulle være instruktør på kurset. Grunnen var i hovedsak at dette er begreper som referer til noe som vi oppfattet som vesentlig for objektorientert forståelse. Samtidig har begrepene definisjoner som i første omgang kan være vanskelige å forstå. Vi presenterte begrepene for resten av gruppen, for å få en godkjenning, før vi laget aktivitetene.

Noen vil sikkert kommentere at vi fokuserte modularisering og gjenbruk, fremfor begreper som for eksempel: Data, metode og superklasse. Modularisering og gjenbruk ansees som vesentlige teknikker i programmering og vi ønsket at studentene skulle erfare dette i et slikt kort kurs.

Når det gjelder vårt valg om ikke å se på uttrykk som data og metode, har dette sin begrunnelse i at vi hadde som forutsetning at studenter som har søkt ett informatikkstudium ved et universitet kjenner begrepet data. Vi skulle bare klargjøre ved hjelp av dialog i starten på kurset, at vi hadde en felles forståelse av begrepet. Metodebegrepet er ikke et spesifikt OOP begrep, men et begrep som i blant annet Java representerer en programbit som utfører en funksjon på data i et objekt.

Det vi gjorde når vi tok i bruk begrepet data var at jeg tegnet Figur 17: Illustrasjon av klassebeskrivelse med objekter. Vi omtalte regnr., farge, vekt og motor som data. Begrepet metode innførte vi ved at vi først snakket om å gi instrukser til roboten.

Først på et senere tidspunkt sammenlignet vi dette uttrykket med begrepet metode. På videoopptakene kan jeg registrere at det skjedde at vi brukte uttrykket operasjoner og funksjonalitet istedenfor metode. Ordet funksjonalitet brukte vi blant annet for å beskrive at et bilobjekt kunne utføre tjenester som for eksempel: ABS-bremser og sjekke at dørene er låst. Jeg vil i ettertid gi uttrykk for at vi her kanskje skulle vært mer konsistent med begrepsbruken og startet med for eksempel begrepet operasjoner før vi innførte metodebegrepet. Dette siden operasjon er et generisk uttrykk som brukes uavhengig av programmeringsspråk, mens metodebegrepet er knyttet til enkelte programmeringsspråk. Dette siste er også en medvirkende årsak til at jeg ikke har metodebegrepet med på listen over de begreper jeg ser nærmere på. Jeg har ikke kunnet registrere noen indikasjoner på at begrepene data og metode var noe problem for studentene.

9.1.1 Objekt-begrepet

Objektorientert programmering er et begrep som bygger på at selve fundamentet er objekter med egenskaper. Egenskaper er data som beskriver en tilstand, det vil si de dataverdier som finnes lagret i objektet i et definert øyeblikk, samt operasjoner (… enkelt handling, el. delhandlinger sett som en enhet: *arbeids-, regneoperasjon /…*; Norsk ordbok) som kan utføres. Disse operasjonene utfører noe som ofte omtales som en hendelse. Disse hendelsene kan bestå av å endre data i eget objekt eller andre objekter som objektet har tilgang til, hendelsen kan også være et kall til en operasjon som finnes i et annet objekt.

I kapittel 8.1 Første dagen, har jeg beskrevet hvordan vi introduserte begrepet objekt for studentene. Vi tok utgangspunkt i at vi ville prøve å binde begrepet til en oppfattelse av den verden som studentene måtte være med å definere. Ved å bygge på de kunnskaper og erfaringer som studentene intuitivt knyttet til begrepet objekt, håpet vi på å få til en tidlig opplevelse hos studentene av at objektorientert tankegang ikke var så annerledes enn det de kjente fra før.

Objektbegrepet ble akseptert nokså direkte, og det at studentene klarte å assosiere til en for dem kjent verden kom også frem av kommentarene. En spurte blant annet om: ”Betyr det at en person og for eksempel en klasse kan være objekt?” Jeg svarte da: ”Jeg er litt usikker på hva du mener med klasse, her i denne sammenhengen?” For ikke å henlede på klassebegrepet stoppet jeg her. Da sier vedkommende: ”En klasse på en skole.” Da fikk jeg anledning til å prate litt om: ”At en person som vi kan identifisere entydig ved hjelp av fødselsnummer eller studentnummer er et objekt. Det samme gjelder for klasse 1A, som kan identifiseres med 1A og et årstall. Men malen som beskriver hvilke data og hendelser objektene skal ha, kalles i programmering for en klasse. Objektet klasse 1A er noe helt annet enn malen som vi kaller klasse.” Under dette tegnet jeg Figur 17: Illustrasjon av klassebeskrivelse med objekter, på tavla. Jeg stilte spørsmålet: ”Hva er da en klasse?”. Svaret kom nokså overraskende: ”En klasse 3A er et objekt, mens flere klasser er en mal.” Nå ble det en diskusjon hvor de klargjorde at flere klasser også er objekter og at de kommer fra en felles mal.

Etter den oppsummerende diskusjonen ble begrepet objekt brukt av studentene i henhold til den beskrivelsen vi var kommet frem til. Studentene slet litt med å huske at programmet måtte beskrive hvilket objekt som skulle utføre noe. I Aktivitet 2 (Vedlegg 3) skulle to objekter ta opp to beepere hver. En av gruppene skrev en kommando til et objekt om å gå et skritt frem med: *karel.move();* og så gjentok de kommandoen på neste linje. Når de kjørte programmet ble de overrasket over at det ene objektet beveget seg to skritt, istedenfor slik de hadde tenkt det, nemlig at hvert av objektene beveget seg et skritt. Den forståelsen som dette kan indikere er at studentene ikke hadde klart for seg at ordet *karel* i kommandoen identifiserte objektet. Dette blir forsterket av kommentaren, som var: ”Beveger den seg to skritt?” Og den andre sa: ”Jeg trodde at den andre skulle også bevege seg.” Dette tyder på at det ikke bare var en forglemmelse i øyeblikket, men at det hadde dannet seg en oppfattelse av kommandoen, uten at de hadde en klar oppfattelse av at ordet Karel identifiserte et objekt.

Senere i kurset opplevde vi at studentene skrev en kommando som et objekt skulle utføre og så registerte de at det var ”feil” objekt som utførte kommandoen. Da var de umiddelbart klar over at de hadde brukt feil navn for å identifisere objektet som skulle utføre den aktuelle kommandoen og rettet i kildekoden.

En illustrasjon på hvordan studentene oppfattet objektene i vårt eksperiment: ”Han må jo flytte seg, han derre Kjakan.” Dette indikerer at objektet er noe de har et forhold til, en form for personifisering. Uttalelsen kan også være begrunnet i frustrasjon, men viser også et engasjement i arbeidet. Vår måte å omtale objektene på kan nok ha ledet til at det var flere lignende uttalelser.

En kommentar vi fikk fra en gruppe under arbeidet med Aktivitet 2 (Vedlegg 3) var: ”Må vi gi kommandoer til objektene annen hver gang, eller kan vi først la det ene objektet ta opp begge sine beepere før vi starter å bevege det andre?” Denne kommentaren kan komme av at når vi snakket om objektene så omtalte vi det slik at vi ba først Karel ta et skritt og så Carol ta et skritt. Dette gjorde vi for å få det mer tydelig at vi måtte identifisere objektene for at programmet skulle vite hvilket objekt som skulle utføre noe i skjermbildet. Men denne måten å legge det frem på, medførte kanskje at studentene oppfattet det slik at det var slik det skulle gjøres. Nå var dette et spørsmål som det var greit å svare på, og som ga oss mulighet til å legge frem en kort introduksjon til modulalisering av kode. Mer om dette senere.

Nå kunne vi registrere, at enkelte ganger ble begrepet brukt feil, men da var det mer feil ord, og ikke forståelsen som var feil. Et eksempel på dette er at vi på videoopptakene kan registrere at de bruker ordet objekt når de egentlig snakker om en klassedeklarasjon. Etter at en gruppe fikk feilmelding på kallet til en metode sa første person: ”Vi har ikke den metoden i objektet”. Så svarer den andre:” Den var jo der i går”.

Her er det jo i og for seg riktig at objektet ikke hadde metoden, men poenget er at de snakker om objekter med samme navn, men generert fra to forskjellige klasser. Dette var noe som skjedde i en del tilfeller at de brukte ordet objekt, men i realiteten refererte til klassedeklarasjonene. Dette kan komme av at vi i starten ikke brukte eksakte definisjoner på begrepene, men at vi fokuserte på at studentene hadde en viss forståelse av begrepene og hvorledes de brukes i objektorientert

programmering. Vi vet også at vi av og til ikke var helt klar i vår språkbruk, for eksempel når vi hadde figurer som både viste klassen og objektene som ble generert fra klassen (Figur 17: Illustrasjon av klasse beskrivelse med objekter). Dette var et bevisst valg fordi vi ønsket at studentene skulle være klar over at det var forståelsen av begrepene og ikke en eksakt definisjon som var vårt fokus. Dersom et slikt opplegg skal gjennomføres for et kurs, vil jeg anbefale at en allerede i starten, bruker litt mer tid på å jobbe med begrepsbruken av objekter og klasse. En måte å gjøre det på kan være å prøve å innføre litt av opplegget i boken "Objects First With JAVA, a practical introduction using BLUEJ", Second edition (Barnes og Kölling, 2005) omtalt i kapittel 5 Vurdering av bøker. Den måten som de introduserer objekt og klassebegrepene på vil være et alternativ for å bygge en klarere innsikt i sammenheng og forskjell på bruken av de to begrepene i programmering.

På kursevalueringen som vi hadde i plenum kom det frem to uttalelser rundt objektbegrepet. Den første var: "Vi slet med at vi kalte en funksjonalitet uten å si hvilket objekt som skulle utføre den." Dette stemmer med det jeg har beskrevet ovenfor. Uttalelsen kan også indikere at studentene hadde en sterkere opplevelse av dette problemet enn vi registerte. Den neste var litt overraskende i det den var på et mer abstrakt plan: "Forskjellen på objektorientert og prosedyreorientert. Det å tenke på objekter, og så på hva de inneholder." Overraskelsen er her at jeg hadde forventet at studentene skulle fokuser mer på problemer de hadde opplevd når de skrev kode og mindre på slike strukturelle begrepsoppfattelser.

9.1.2 Klasse-begrepet

Vår tanke var å gi studentene først en oppfattelse av begrepet objekt, pluss litt erfaring i å observere et objekt på skjermen, før vi introduserte klassebegrepet. Vi skulle bare omtale at vi hadde en mal som ble brukt for å generere objektene. Men som jeg har beskrevet tidligere brukte en student ordet klasse helt i starten og det ble til at vi innførte begrepet tidligere enn vi hadde planlagt. Nå ga vi istedenfor det planlagte, en forklaring som knyttet begrepet klasse til begrepet mal. Det vil si en mal som en lager gjenstander etter. Vi kunne jo på det tidspunkt unngått å korrigere, men formuleringen som studenten brukte: "Flere klasser er en mal", ville ikke stemme med den forklaringen de senere skulle få. Vi ville unngå å introdusere feil beskrivelse, selv om vi ville akseptere unøyaktig bruk i starten. Det ble derfor naturlig at vi presenterte en kort forklaring om begrepet klasse slik det er beskrevet under temaet objekt.

Når studentene startet med aktivitetene foretrakk de å bruke begrepet mal istedenfor klasse. Dette kan komme av at vi oppmuntret til det, ved at vi i størst mulig grad prøvde å snakke om mal. Men for å få til en overgang til begrepet klasse sa vi enkelte ganger: "..mal, eller klassebeskrivelsen…" Under Aktivitet 1 og Aktivitet 2 ble objektene generert fra samme mal og vi kunne ikke registrere noen tilfeller av feil bruk av begrepene, utenom at de foretrakk ordet mal istedenfor klasse. Her er det en svakhet at vi ikke har video fra disse første aktivitetene, derfor er det ikke noen dokumentasjon av hva studentene snakket om og gjorde når de ikke ble observert.

Overgangen fra ordet mal til ordet klasse viste seg for selve begrepet klasse å gå greit for studentene. Men derimot ble det en del frustrasjon av forståelsen av hvilken klasse som hadde hvilke egenskaper. I KarelJ systemet vårt, fantes det flere predefinerte robotklasser som hadde forskjellige egenskaper, for eksempel: Robotobjekter fra en klasse kunne utføre *karel.turnRight();* mens objekter fra en annen klasse måtte programmeres med tre *karel.turnLeft();* for å utføre de samme bevegelsene. En vesentlig årsak til at studentene slet litt med dette var kanskje at navn på klassene til en forveksling kunne ligne på hverandre. Eksempler: ur_Robot, ur_Robot2, Robot og lignende. Det var nok en klar svakhet i opplegget, at vi hadde flere klasse og objektnavn som til en forveksling var lik.

At disse navnene er så like, førte til frustrasjon som svekket mestringsfølelsen siden studentene opplevde at de fikk feilmeldinger på metodekall som de tidligere har brukt. Årsaken til problemet var at de tidligere hadde brukt samme metodekall, men ikke husket at de da hadde generert robotobjekt fra en annen klasse. Problemet kan ha blitt forsterket av at identifikasjonsnavn på objektet de benyttet, var det samme i begge tilfellene. Disse opplevelsene svekker også læring, idet studenten i en tidlig fase opplever inkonsistens i den kunnskapen som konstrueres. På den annen side gir det en sterkere følelse av mestring når studentene etter en tid klarer å håndtere kompleksiteten rundt klassenavn, metodenavn og objektnavn. Det støtter også opp om teorien og målene for PBL (4.4 Problembasert læring) slik den er beskrevet tidligere, idet studenten selv må tilegner seg innsikt i hvorledes slike navneproblem kan håndteres i en programmeringsverden som krever stor nøyaktighet med hensyn på programkoden.

Under arbeidet med Aktivitet 4.2 (Vedlegg 3) hadde jeg en dialog med en gruppe, og en student sa: ”Vi vet at klassen har mange egenskaper som vi ikke ser, men hvordan? Jeg skjønner det ikke helt.” Denne uttalelsen trodde jeg først kom av samme problem som er beskrevet ovenfor. Men i diskusjonen ble det avklart at spørsmålet dreide seg om hvorledes de som hadde laget KarelJ systemet, for eksempel: ”Får roboten til å stoppe når den kolliderer med veggen.” Vi ser her at gruppens problem ikke var å velge riktig klasse for å generere objekter med, men å forstå hvorledes systemutvikleren skriver koden som gjør at systemet virker slik det gjør. Eksemplet illustrerer iver etter å få mer innsikt i hvorledes mer kompleks programmering gjøres.

I den etterfølgende diskusjonen kom det frem at studenten klarte å abstrahere og generalisere problemet sitt, til også å gjelde bruk av andre datasystemer, som for eksempel tekstbehandleren.

I oppsummeringen ved kursevalueringen stilte jeg følgende spørsmål: ”Hva er en klasse?” Første kommentar var: ”På en måte malen for objektene. For hva objektene skal inneholde…” Svaret tyder på at de har en innsikt i hva klassebeskrivelsene brukes til i programmering. Personlig liker jeg den andre som var: ”Beskriver hvilke egenskaper objektene skal ha.” Jeg synes den er beskrivende og nøyaktig nok til å kunne brukes i undervisningssammenheng, idet den omfatter både data og operasjoner.

9.1.3 Sub-klasse-begrepet

Temaet sub-klasser regnes som sentralt i objektorientert programmering og det er derfor litt rart at temaet behandles nokså sent i en del lærebøker (5 Vurdering av bøker). Eksempler fra dette kapitlet er Java Gently (Bishop, 1998), som introduserer at det er behov for sub-klasser i starten, men stoffet som omtaler temaet først står på side 267. Selv i boken Objects First With JAVA, a practical introduction using BLUEJ, Second edition (Barnes og Kölling, 2005) kommer temaet først på side 217.

Når vi allerede i Aktivitet 3 (Vedlegg 3) introduserer temaet kan det vel sies å være tidlig. Studentene hadde allerede på dette tidspunkt erfart at det kunne bli mye programmering når de skulle snu til høyre, siden de måtte gjenta *turnLeft();* tre ganger for å få det til. Når vi da ved hjelp av figur på tavla beskrev hvordan en sub-klasse arver egenskapene fra superklassen, pluss at de i tillegg kan få egne spesielle egenskaper, gav studentene uttrykk for at det var greit.

Under diskusjonen om sub-klasser sa en av studentene: ”Er det slik å forstå at vi ved å lage en *turnRight();* så lager vi et nytt lag mellom ur-Robot og programmet vi skriver?” Dette kan indikere at studenten hadde dannet seg et bilde av at den nye klassen hentet nødvendige tjenester fra superklassen for å få *turnRight();* til å fungere, heller enn at den nye klassen inneholdt alle tjenester.

Uttalelsen er ikke en kopi av noe vi sa, men konstruert på kunnskap om lag-modellen (7.3.3 Starten på kurset) og vår beskrivelse av arv. Dette kan forhåpentligvis tyde på at studenten har utviklet en kunnskap som kan overføres til andre lignende problem de støter på i programmering.

Uttalelsen brukte vi for å få frem en umiddelbar og positiv forsterking av læring slik Jens Kaasbøll(2002) beskriver det. Dette gjorde vi ved at vi refererte uttalelsen og prøvde å utdype den ved hjelp av positive kommentarer i den videre diskusjon.

Når studentene skulle komme i gang med arbeidet med sub-klasser, kunne vi registrere at studentene fikk problem med konstruktøren. De hadde vanskelig for å se hvorfor de skulle bruke konstruktør og mente det var unødvendig og komplisert og argumenterte for måter å komme rundt dette. De prøvde med å si at vi kunne ha flere klasser på samme nivå, ikke bruke arv og forskjellige forslag til utforming av kildekode. Siden disse eksemplene ikke er sentrale i mitt tema tar jeg ikke utdyping av eksemplene. Etter en diskusjon godtok de at i Java må vi bruke konstruktørene slik vi beskrev. Eksemplet utdypes nærmere i 9.3.1 Situasjonsbetinget og tre spørsmål.

Videre i kurset registrerte vi svært lite problemer knyttet til begrepene sub-klasser og arv. Det som dukket opp var i hovedsak knyttet til problemer i forbindelse med det som tidligere er omtalt som likhet i navn på klasser.

Det kan sies at vi brukte enkle eksempler og at det ikke er påvist noe om hvorledes kunnskap om mer komplekse konstruksjoner med sub-klasser vil absorberes av studentene. Det vil derfor være interessant å prøve ut dette ved en senere anledning.

9.1.4 Modularisering-begrepet

Dette begrepet defineres av Barnes og Kölling (2005) slik: ”**Modularization** is the process of dividing a whole into well-definded parts, which can be built and examined separately, and which interact in well-definded ways.” Dette er en beskrivelse av modularitet som retter seg mot det som i Java heter pakker, klasser og metoder. Det å bygge denne kunnskapen hos studenter som er i ferd med å starte på et innføringskurs i programmering ser jeg for meg vil være vanskelig. Men jeg ville gjerne introdusere studentene til begrepet modularisering. Dette ønsket jeg å gjøre ved å anvende begrepet med en litt mer utvidet definisjon, enn definisjonen som er beskrevet ovenfor (Barnes og Kölling, 2005).

Vi forklarte begrepet modularisering for studentene med: Det å dele programkode opp i deler, og da deler som en som skriver programkode kan oppleve som en logisk enhet. En modul kan gjerne inngå som en del av en større modul.

Denne definisjonen bruker jeg på to måter, den ene er om en programkodedel som i arbeidssammenheng kan fungere som en logisk del med hensyn på testing for syntaksfeil og sjekk av at semantikken stemmer med intensjonen til programmereren. Siden den første måten ofte tar for seg små programbiter, er den å betrakte som et mikroperspektiv i forhold til den andre måten, Den andre måten er mer å betrakte som en videreutvikling av den første. For å beskrive den, bruker jeg et annet sitat fra Barnes og Kölling (2005): “To help us maintain an overview in complex programs, we try to identify subcomponents that we can program as independent entities. … In object oriented programming these components and subcomponents are objects.” Der spesielt den siste setningen bruker et makroperspektiv på begrepet.

Årsaken til at jeg bruker to måter, har som bakgrunn at den første hevder jeg er mer egnet for å introdusere begrepet modularisering i et introduksjonskurs i programmering. Dette fordi det vil gi studentene trening i å bruke moduler som inneholder få linjer kode. Derfor kan dette lett stimulere mestringsfølelsen (4.2 Mestre), og studentene kan erfare å jobbe med moduler før kompleksiteten i programmeringsoppgavene øker, jamfør uttalelsen til Barnes og Kölling (2005) ovenfor. Det kan sies at min beskrivelse ikke er konsistent idet den har overlappende beskrivelser. Jeg skal utdype modulbegrepet nærmere med et eksempel på en modul som en gruppe skrev og så testet:

```
Karel.move();
While (Karel.frontIsClear()) {
        Karel.move();
}
Kalrel.turnLeft();
Karel.move();
```

Figur 31: Modul fra en gruppe

Her beveger koden roboten til veggen treffes, så snur den til venstre og tar et skritt frem. Slik jeg ser det, er modulens størrelse bestemt ut i fra hva studentene fant som et passende antall linjer med kode, når de skulle teste syntaks og semantikk. Mens det for en litt mer erfaren programmerer kan være mer naturlig å skrive en kortere modul, som kan være hele while setningen, teste at den fungerer og så la denne delen inngå som en del i en mer omfattende modul.

Modularisering var det begrepet som det i første omgang virket som studentene forstod og som de uttrykte var: "..helt greit, ". Men det viste seg at det var den begrepsforståelsen som hadde mest sprik når de skulle prøve å la begrepet bli reflektert i sitt arbeid. Dersom jeg tar utgangspunkt i de definisjonene som er beskrevet tidligere, er begrepet atskillig mindre presist enn objekt, klasse og andre kompilerbare begreper. Det kan virke som studentene oppfattet objekter som moduler. Dette er for så vidt riktig, men studentene hadde vanskeligheter med å innføre modulær tenking i arbeidet med programkoden. Med dette mener jeg at de overraskende lenge brukte et mikroperspektiv på begrepet. Det kan være at den upresise beskrivelsen av begrepet medførte større usikkerhet hos studentene enn vi forventet. Denne forskjellen i hva vi observerte og hva vi forventet, kan indikere at desto mer kompleks definisjoner er, desto mer erfaring kreves for å la begrepsanvendelsen gjenspeiles i programmeringen. At studentene i starten av kurset hadde vanskeligheter med å produsere det en erfaren programmerer omtaler som modulært, kan være naturlig i et slikt introduksjonskurs. Da med tanke på at de etter hvert erfarer flere varianter av moduler, og dermed finner ut hva som kan være mest hensiktsmessig i den enkelte situasjon.

Under kapitlet om objektbegrepet beskrev jeg en uttalelse knyttet til Aktivitet 2: " Må vi gi kommandoer til objektene annen hver gang, eller kan vi først la det ene objektet ta opp begge sine beepere før vi starter å bevege det andre?" Uttalelsen kan ha kommet av at studenten tenkte på moduler og mulighet for å modularisere programmet, eller var det bare et ønske om i større grad å kunne bruke "klipp og lim"? En av gruppene startet med å la objektene bevege seg annenhver gang, men etter en tid, delte de opp koden slik at det ene objektet gjorde seg ferdig før det andre startet. Dette kan komme av at de ville prøve denne varianten, eller at de så at det ville være enklere å lete etter feil i programmet dersom koden som styrte et robotobjekt var samlet.

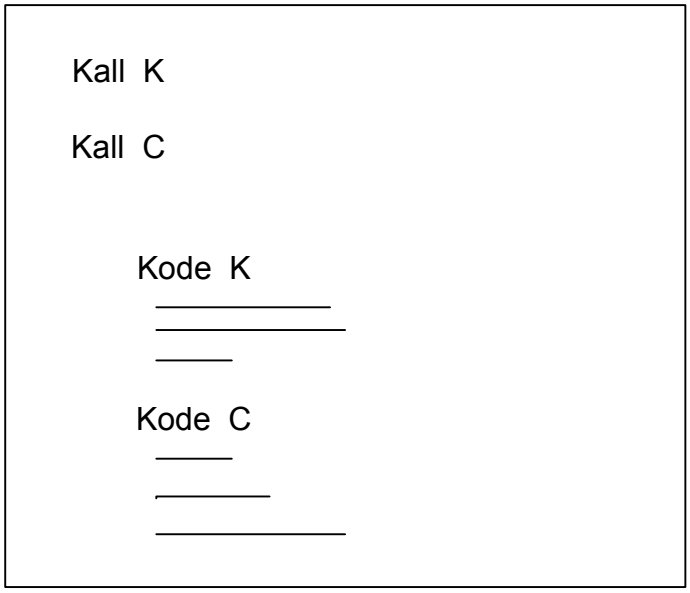
Denne gruppen var den gruppen som kom raskest i gang med å bruke moduler i sitt arbeid, i henhold til min definisjon av begrepet. Med dette mener jeg at de skrev en programmodul som de fant logisk og kompilerte. Denne formen å arbeide på gjorde at de fikk få linjer symbolkode å forholde seg til, og de klarte dermed raskt å rette syntaksfeil som kompileringen avdekket. Ved kjøring av programmet og avdekking av semantikkfeil var det nok litt hjelp i at robotobjektet gjorde enkle bevegelser som rett frem, forandre retning mellom hver test. Derfor klarte de enkelt å se om bevegelsen stemte med det som var intensjonen med koden de hadde skrevet.

Det var særlig når gruppen jobbet med de første aktivitetene at de logiske biter var korte. Når de kom i gang med Aktivitet 5 (Vedlegg 3) ble bitene lengre og mer kompliserte. Kanskje følte de seg sikkrere og derfor ville skrive mer kode før de testet? I Aktivitet 4-2 (Vedlegg 3) skulle studentene analysere problemet ved hjelp

av papir før de startet. Denne gruppen beskrev for meg et løsningsforslag samtidig som de henviste til kartet. Forslaget var oppdelt i moduler ut i fra bevegelsene til roboten. Gruppen holdt seg til modulene når de skrev koden, men de prøvde å skrive mer kode mellom hver test. Dermed fikk de problem med at robotobjektet ikke beveget seg slik de håpet den skulle gjøre og de slet de med å finne feilen i koden. Etter hvert gikk de tilbake til å teste mindre deler. Nå gikk det greit å få koden slik at objektet gjorde det de ønsket (6.6 Konstruktivistisk). Senere i kurset brukte gruppen samme metode, å dele opp i mindre deler når de fikk problemer med koden.

Den andre gruppen delte i liten grad opp koden i moduler. Det vi oppfattet første dagen viste seg å stemme overens med det vi ser av videoopptakene fra resten av kurset. Koden de laget hadde ikke så klare logiske moduler knyttet til robotens bevegelser som den første gruppen hadde. Det er mulig at studentene oppfattet at de hadde en logisk deling knyttet til robotbevegelser, men vi kunne ikke observere dette. De hadde mer en arbeidsform som kanskje kan sammenlignes med ”situated actions” beskrevet i ”Plans and situated actions” (Suchman, 1994). Med dette mener jeg at de skrev kode, prøvde, og fortsatte å legge til ny kode uten at de prøvde å se den nye delen som en del av en helhet. De fikk veiledning og tips om hvorledes de kunne modularisere koden og det ble en utvikling i retning av mer mudularisering. Men gruppens syn kan illustres av kommentaren jeg fikk når de arbeidet med Aktivitet 4-2.

”Vi tar og prøver, bare ser åssen det går. Nei, f.. det tar jo en evighet jo, kan ikke følge opp anmodinga din om å kjøre kompilering ofte, når det tar 5 år å, he, he”. Kommentaren indikerer klart at det å vente på at kjøringen av programmet skal komme frem til det sted i koden de ønsket i prøve, tar lang tid. Den indikerer også at de ikke er klar over hvorledes de ved en tydeligere modularisering av koden enkelt kan kommentere bort deler av koden for raskere å få kjøringen frem til den delen som skal prøves. Etter kommentaren viste jeg prinsippet for å kommentere bort kall ved hjelp av å tegne figuren:



Figur 32: Prinsipp for å kommentere bort

Mine kommentarer var ganske korte: ”Ved å ha det som er spesielt for Kall K i (peker på Kall K og så på Kode K) kan jeg bare kommentere bort (peker på Kall K), for å
spare tid”. De tok poenget og endret koden slik at selve testkjøringen gikk raskere.

Frem til dette tidspunktet hadde gruppene vært nokså jevne med hensyn på produksjonshastighet av kode når de jobbet med aktivitetene. Det var først ved Aktivitet 4-2 (Vedlegg 3) at det ble en merkbar forskjell mellom gruppene i tidsforbruket før en aktivitet var klar. Gruppen som brukte minst modularisering i koden brukte klart lengre tid på denne aktiviteten. Årsaken til dette kan være at de frem til dette tidspunktet var fornøyd med sin kunnskap med bakgrunn i at de mestret å lage løsninger. Derfor kan de ha manglet nok motivasjon for i sterkere grad å bruke moduler i koden. Når de på dette tidspunktet fikk en oppgave som var mer kompleks, og at de fortsatte å skrive lengre kodebiter før de kompilerte og testet, kan ha bidratt til at de brukte mer tid.

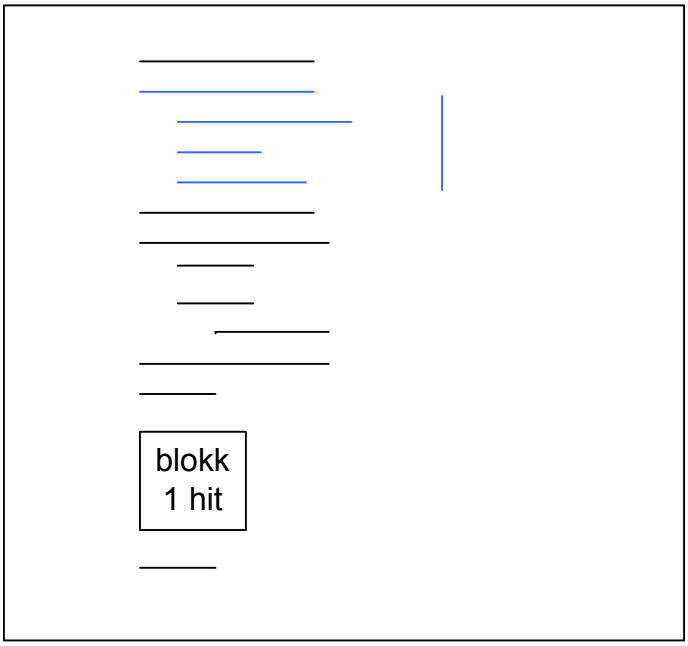
Om dette kommer av kompleksiteten eller av lengden på kodebitene, er vanskelig å si noe om, siden kompleksiteten kan bidra til at kodebitene blir lengre. Det kan også være det at de aktivitetene vi brukte ikke i tilstrekkelig grad klarte å bidra til en motivasjon hos studentene med hensyn på å anvende moduler i programutviklingen. Forskjellen mellom gruppene er en indikasjon på at det å konstruere kunnskap kan kreve ulik mengde arbeid. Derfor burde vi kanskje ha hatt aktiviteter som på et tidligere tidspunkt la opp til å få en mer tydelig erfaring med fordeler med en mer modulbasert programmering. Dette berører nettopp en av hovedproblemene ved introduksjonskurs, dilemmaet ved å prioritere temaer.

Som jeg har beskrevet tidligere i kapitlet, så bidro studentenes interaksjon med datasystemene til at begreper som objekt og klasse raskt kunne anvendes i programmeringen. Tar vi derimot utgangspunkt i min beskrivelse av at studentene hadde problemer med å anvende modularisering i sin programmering, er begrepet et godt eksempel på at overføringsverdien av muntlige råd fra undervisningspersonalet antagelig har mindre læringseffekt. Spesielt når det gjelder den type kunnskap som undervisningspersonalet prøver å overføre som: Gode råd og smart å gjøre.

Etter hvert gikk det ganske bra med hensyn på å anvende moduler i programmeringen, og da i et mikroperspektiv. Det kan være flere årsaker til dette. En som jeg hevder er sentral, er at studentene hadde god tilgang på veiledningsressurser som kunne påpeke og kommentere dette i aktuelle situasjoner. Slik tilgang kan være et større problem i et tradisjonelt kurs på universitetet. En annen kan være er at de etter hvert følte at de lykkes bedre ved å anvende modularisering. Dette gjelder særlig den gruppen som er omtalt som den første gruppen i kapittel 8 Gjennomføring av eksperimentet. Det samme gjelder til en viss grad også den andre gruppen. Men denne gruppen er et eksempel på at selv om deltakerne viste innsikt i begrepet modularisering, leder det nødvendigvis ikke til at de samme studentene lager strukturert kode. Som det står beskrevet i det samme kapitlet, var koden deres meget ustrukturert og klart et resultat av utvikling etter prinsippet: ”Situated actions” slik det er beskrevet tidligere i dette kapitlet (Suchman,1994). Det ville vært av interesse å se nærmere på om ikke litt endring i oppgavenes formulering kunne hjulpet for å få en forbedring på dette.

9.1.5 Gjenbruk-begrepet

Under Aktivitet 2 (Vedlegg 3) fikk vi som tidligere beskrevet i kapittel 8.1 Første dagen, et spørsmål som vi brukte for å introdusere modularisering av kode. Under dette brukte vi ord som: Deler av kode, kodebit, modul, gjenbruk, objekter kan beveges uavhengig av hverandre, samtidig som vi illustrerte på tavlen med figur:



Figur 33: Strekkode og blokker som kan kopieres

Til figuren er det å bemerke at det blå viser hvilken del av koden jeg pekte på, for så å illustrere at jeg flyttet den ned til ”blokk 1 hit”.

Vi brukte figuren ved å peke og illustrere flytting med hendene. Forklaringen til figuren var, at dersom studentene programmerte først det ene objektet, og det andre skulle gjøre bevegelser som lignet, var det mulig å kopiere koden. Så var det bare å endre navnet som identifiserte det nye objektet, samt endre de delene av koden som skulle gi andre bevegelser enn det første objektet hadde.

Det kan kritiseres at vi brukte strek på tavlen istedenfor å bruke reel kode. Men vår hensikt var her å vektlegge prinsippet og unngå at studentene skulle henge seg for mye opp i selve kodeteksten i eksemplet. Som vi ser i første avsnitt brukte vi mange ord. I ettertid kan jeg se at vår iver etter å bruke ord som studentene kunne assosiere, i enkelte situasjoner kan forvirre enkelte studenter. Jeg er i ettertid av den oppfattning at i den situasjonen hadde det holdt med et uttrykk, for eksempel: Del av kode.

Gruppen som hadde tatt opp spørsmålet, løste aktiviteten ved at de først gjorde ferdig det ene robotobjektets bevegelser, før de startet å skrive programdelen for det andre robotobjektet. Dette tolker jeg som at de brukte modularisering, i det de delte opp koden i moduler, men siden de skrev koden på nytt klarte de ikke å utnytte muligheten for gjenbruk av kode. Dette viser at disse studentene på det tidspunkt ikke så noen nær knytning mellom disse begrepene. Eller at de kanskje bare ikke hadde nok erfaring til å huske muligheten for gjenbruk.

Den gruppen som var flinkest til å bruke moduler ble etter hvert flinkere til å kopiere kodedeler og bruke de andre steder i samme programmet, men mens de strevde med Aktivitet 4.2 hintet vi dem om at de kanskje hadde gjort noe lignende i en tidligere aktivitet. Etter en kort diskusjon ble de klar over hvilken metode de kunne anvende på nytt. De lette frem det aktuelle programmet og kopierte metoden inn i det nye programmet. Da det virket, kom kommentaren: ”At vi ikke har gjort det før.”

Dette kan være et eksempel på at det kanskje går tid før beskrivelser fra undervisningspersonalet blir kunnskap for studenten. Vi hadde jo tross alt på det tidspunktet sagt og vist på tavlen et antall ganger, argumentasjon for at det kunne være lurt å anvende gjenbruk av programkode. Se også omtale av ”tips og råd” under kapittel 9.1.4 Modularisering-begrepet. Men det kan også være at studentene først på det tidspunkt fikk en egenerfaring som var sterk nok til at utsagnet kom.

Ser vi på disse utsagnene og bruker definisjonen til Glanz et al,(2002) “Health Behavior and Health Education. Theory, Research and Practice.” på mestring slik den er beskrevet i kapittel 4.2 Mestre, kan det indikere at årsaken til uttalelsen var at de på et tidligere tidspunkt ikke mestret deloppgaver som er nyttig for å gjenbruke moduler. Eller at vi ikke hadde gitt det nødvendige bidrag slik at de kunne bygge kunnskap før vårt hint kom som en utløsende faktor.

Selv tror jeg ikke at vårt hint var noe vesentlig bidrag. Det var heller det at de erfarte at dette kunne være et positivt bidrag til å produsere en raskere løsning på problemet. Dette er det som Laurillard (1993) omtaler som direkte læring ved hjelp av persepsjon, og Kaasbøll (2002) betegner som positiv og umiddelbar forsterking.

9.2 Betydningen av didaktiske elementer

Foran har jeg tatt opp hvorledes studentene bygget en begrepsforståelse for et utvalg av objektorienterte begreper. Her skal jeg drøfte noen elementer som kan ha gitt bidrag til byggingen av begrepsforståelsen. Først tar jeg noen generelle betraktninger før jeg skal prøve å se på de enkelte element og kombinasjoner av disse.

Vi la opp til at studentene skulle konstruere sin egen kunnskap gjennom aktiviteter og diskusjoner. Det at vi la opp til at kunnskapen skulle konstrueres, betyr ikke nødvendigvis at kunnskap de hadde ervervet seg var like lett å hente frem igjen på et senere tidspunkt. Vi erfarte under kurset at de brukte begreper feil, selv om de tidligere hadde brukt begrepet på en tilfredsstillende måte. Dette kan komme av at

de ikke hadde en klar oppfattelse av begrepet eller rett og slett at de ikke husket det i farten.

Vi observerte at studentene laget forslag til løsninger som ikke virket slik de trodde de skulle, og at dette kom av at de brukte feil instruksjon eller feil kombinasjon av programsetninger. Et annen illustrasjon på usikkerhet i når og hvordan kunnskapen skulle brukes, er spørsmålene de stilte, for eksempel: ”Kan vi bruke, if vegg in front then turn right?” Her er de ikke sikker, og ber om bekreftelse på at de tenker riktig. Det kan indikere at de ikke er sikker på at deres forståelse er lik vår forståelse. Det var litt overraskende at tilsvarende usikkerhet på forståelse skulle komme frem så pass ofte under kurset.

Observasjonene indikerer også at dess flere repetisjoner og tid til bearbeiding som studentene fikk, jo lettere hadde de for å hente frem kunnskapen. Jeg vil nok i ettertid si at jeg undervurderte at studentenes behov for repetisjoner er en viktig ingrediens for å bygge kunnskap.

9.2.1 PBL og parprogrammering

PBL som utdanningssystem gav et godt bidrag ved at noe av poenget med PBL er å la studentene i større omfang få ta del i sin egen utvikling av kunnskap, ved hjelp av å finne en løsning som kan fungere (Pedagogisk forskningsinstitutt ved Universitetet i Oslo).

Det kan sies at den enkelte aktivitet i sommerkurset var små sammenlignet med de oppgavetypene som beskrives i PBL sammenheng på nettsidene til universiteter på nettet. Men det at vi hadde ett domene (Karels verden), som hoveddelen av programmeringen skulle foregå i, og at jeg hevder at vi fulgte samme didaktikk, gjør at kurset kan sammenlignes med en del av de mindre oppgavene som er beskrevet på nettet. Likheten består i at vi hadde grupper som ble presentert et problem, så måtte deltagerne selv finne forslag til løsning. Når de prøvde ut forslagene ville de treffe på problem som ikke tidligere var beskrevet, og de måtte foreslå løsning på disse.

Siden jeg ikke har noen referanseundersøkelse å vise til, kan jeg ikke si om andre undervisningsopplegg ville ha gitt samme eller bedre resultat. Det vi vet er at PBL har gitt gode resultat ved flere universitet når det gjelder å bygge kunnskap i ferdighetsfag som ingeniørfag, helseutdanning, introduksjonskurs innenfor vitenskapelige fag som matematikk og programmering (4.4 Problembasert læring). Det som ofte fremheves i sammenligning med metoder basert på forelesninger er: God basis for videre læring, problemløsing og aktiv deltagelse i arbeidet.

Det kan se ut som om PBL som metode kanskje passer godt sammen med parprogrammering. Dette siden begge metoder forutsetter at gruppene skal løse arbeidsoppgaven i felleskap. En ulikhet mellom metodene er antallet i gruppene. Siden vi hadde en gruppestørrelse på to, har jeg vanskelig med å klargjøre i hvilken grad det var PBL som metode eller parprogrammering som metode, som bidro til å

skape den settingen som studentene beskrev som et vesentlig moment for å bygge den begrepsforståelsen som jeg har registrert.

Det at vi brukte så små grupper som parprogrammering gir, skulle gi et bidrag til at studentene følte at det var enklere å kommunisere i en slik gruppe enn i en på opptil 8 stykker slik det tradisjonelt brukes ved PBL. Her kommer det inn at veilederne skulle være aktive og kunne delta i gruppene på eget initiativ. Dette prinsippet kan bryte med beskrivelsen til Bereiter og Scardamalia (in press), om at det i PBL skulle være veiledning, men lite hjelp av instruktør.

Det finnes en betydelig dokumentasjon på at hver for seg gir både PBL og parprogrammering et bidrag til forståelse og innsikt i undervisningssammenheng (4.4 Problembasert læring og 4.5 Parprogrammering). Jeg vil hevde at kombinasjonen av disse to metodene bidro til å forsterke resultatet. Jeg har ikke funnet noe materiale som ser på dette og det vil derfor være interessant å se nærmere på i hvilken grad kombinasjonen er komplementær.

9.2.2 Skriftlig materiale

Det skriftlige materialet vi laget skulle være kort, enkelt å lese og støtte opp om arbeidet med aktivitetene (7.2.2 Skriftlig dokumentasjon skulle være kort). Det vil si, bare være støtte til det som forgikk i den muntlige kommunikasjonen i kurset. Dette gjorde at vi kanskje ikke la nok til rette for at studentene skulle få trening og motivasjon for å bygge den type kunnskap som behøves for å drive programmering fra et skriftlig materiale. Behovet vil antagelig bli større med mange deltakere på et kurs. På den annen side er den muntlige kommunikasjonen sentral i opplegget, med begrunnelse i den dynamiske tilpassing til studentenes erfaringsbakgrunn. Videre vil behovet for å situasjonstilpasse det som blir sagt, gjøre at det skriftlige materialet må tilpasses denne formen å undervise på.

Et eksempel på en type litteratur som jeg hevder kan være et godt utgangspunkt for et slikt skriftlig materiale er boken BlueJ (5.3 Java BlueJ) av Barnes og Kölling (2005). De legger opp til at begrepene introduseres ved at studenten ved hjelp av en visualisering på skjermen generer for eksempel objekter. På samme måte kan en lage klasser og instanser (objekter) av denne. Boken legger opp til at studenten skal bygge kunnskap på å erfare gjennom aktiviteter.

Tilbakemeldingene fra studentene var at de var meget fornøyd med at de slapp å lese så mye og at aktivitetene var klare og tydelige slik at de enkelt oppfattet hva som var problemet. De sa også at formen egnet seg godt som støtte til diskusjonene vi hadde. Jeg oppfatter at begrepet diskusjon her omfatter det vi gjorde i innledning før hver aktivitet, veiledning underveis og oppsummeringen. Med unntak av arkene som beskrev aktivitetene, viste det seg at studentene bare brukte det skriftlige materialet når de opplevde at de hadde behov for å finne noe for å løse et problem de jobbet med. Det vil si at jeg ikke kan finne noe i mitt materiale som indikerer at de leste for å sette seg inn i stoffet eller få kunnskap utover det som kunne knyttes direkte til løsning av et problem ved aktivitetene. Dette at studentene rasjonaliserer så sterkt kan være en svakhet idet de kanskje ikke får nok trening i å forholde seg til

den type eksakte definisjoner som ofte finnes i litteraturen. En annen konsekvens vil være at det som skal læres må komme frem via aktiviteter. Kanskje aktivitetene må utformes slik at de stimulerer til at studentene føler behov for støttelitteratur?

9.2.3 Antall deltagere

Med det lille antall deltagere vi hadde, så er det nok en svakhet i datagrunnlaget når det gjelder å kunne vurdere kursoppleggets potensial for oppskalering til store deltagerantall. Med så få deltagere kan det hevdes at vårt kurs var et småskalakurs hvor resultatene ikke uten videre kan skaleres opp til å gjelde kurs med det som Shipman og Duch, (2001) omtaler som storklasse med 240. Basser Department of Computer Science ved University of Sidney i Australia, har gjennomført evaluering av innføring av PBL. Rapporten er en departemental evaluering (Greening, Kay og Kingston, 1997) av et innføringskurs i programmering, og innholder en modell for oppskalering til 500 - 600 studenter. Denne rapporten er klar i sine konklusjoner. PBL baserte kurs har ikke høyere kostnader for gjennomføringen enn det de kaller konvensjonelle kurs har. Det er verdt å merke seg at deres beregninger har med administrative kostnader. Det at vi brukte parprogrammering vil lett kunne bli kritisert for å bli kostbart i kurs med mange deltagere, idet det lett fører til at antall veiledere må økes.

Jeg må erkjenne at vi hadde veiledningsressurser per student, som kan være svært vanskelig å få tilgang til på ordinære kurs ved universitet eller høgskole. Men det at vi blant annet brukte grupper på 2 stykker var for oss et vesentlig poeng slik det er beskrevet i kapittel 6 Begrunnelse for eksperimentet. Det å øke gruppestørrelsen til for eksempel 3- 4 vil gjøre det lettere for studentene å bli passive deltakere og da er mye av vitsen med parprogrammering borte slik det er beskrevet i kapittel 4.5 Parprogrammering.

Tiden vi veiledere brukte på veiledning ble mindre utover i kurset. At behovet for veiledning sank kan tilskrives at studentene fikk mer erfaring i å programmere eller at de tenkte mer gjennom problemet før de ba om hjelp. Men det kan også indikere at ved å ha tilgang til betydelig veiledningsressurser i starten, så vil behovet synke nokså raskt til et lavere behov. Dette fordi studentene da kan få et bedre fundament til å jobbe mer selvstendig med oppgaver.

Observasjonene viser også at når studentene jobbet med Robocode, økte behovet for veiledning slik at det ble betydelig større enn i de andre delene av kurset. Dette tolker jeg som at kunnskapen som var ervervet så langt i kurset ikke var tilpasset nivået i programmeringskunnskaper som Robocode krever.

Men det at vi hadde få deltagere, gjør at vi hadde mulighet for en mer grundig observasjon av begreps- og ferdighetslæringen hos den enkelte gruppe. Jeg fikk prøvd veiledningsmetodikken og via videoopptakene får jeg også innsikt i veiledningen som andre veiledere ga til gruppene. At antall deltakere var så få gjør at vårt utvalg kan være skjevt statistisk relativt den populasjonen som skulle begynne på det ordinære innføringskurset i programmering ved Universitetet i Oslo. Dette forsterkes av at deltakerne selv valgte om de ville delta eller ikke, og det kan

da være at de ikke representerer et randomisert utvalg. Det kan derfor hevdes at mitt resultat ikke har sterk ekstern validitet (3.7 Innsamling av data). Men jeg vil hevde at mitt arbeid kan danne grunnlag for en undersøkelse som kan gi bedre intern og ekstern validitet med hensyn på et statistisk materiale.

9.3 Veiledning og studentdrevet progresjon

I denne tredje delen ser jeg på mine to hovedtema, veiledning og studentdrevet progresjon. I kapittel 2.7 Veiledning har jeg beskrevet min bruk av begrepet veiledning. Vårt didaktiske opplegg med å basere oss på veiledningsbasert støtte til læring gjør at veiledningen kan være et vesentlig virkemiddel for studentdrevet progresjon. Dette er grunnen til at jeg har begge hovedtema i samme kapittel.

9.3.1 Situasjonsbetinget og de tre spørsmål

Når det gjelder veiledning så er det to elementer jeg fokuserer på i denne oppgaven. For det første at den skulle være situasjonsbetinget, det vil her si mest mulig tilpasset studentens kunnskap og erfaring (2.7 Veiledning). For det andre om veiledning basert på de tre spørsmål (4.6 Veiledning baseres på tre spørsmål) kunne bidra til læring av objektorienterte begreper (9.1 Drøfting av læring av OOP begreper).

Situasjonsbetinget veiledning

Situasjonsbetinget veiledning betydde i vår bruk å ta utgangspunkt i at studentene var innehaver av et begrepsapparat når de møtte opp til kurset. Dette betyr at vi ikke kunne ta utgangspunkt i at studentene hadde en felles plattform med hensyn på begreper som det kunne være aktuelt å bruke i forbindelse med kurset. Studentene kan ha vokst opp i andre nasjoners kulturer (2.4 Erfaringsbakgrunn) og dermed også har en annen språklig plattform (2.3 Språk) enn flertallet i studentgruppen har. Men som omtalt i kapitlene kan det selv i flertallsgruppen være forskjell i erfaringsbakgrunn og språk.

Det å presentere et såpass sentralt begrep som objekt, å basere det på studentenes forestillingsverden kan være risikabelt. Et eksempel på det er hentet fra vår innføring av begrepet objekt i kapittel 9.1.1 Objekt-begrepet. Der har jeg beskrevet et eksempel hvor en student kom med en overraskende uttallelse ved å bruke ordet klasse, og vedkommendes oppfattelse av klasse var ikke helt i overensstemmelse med slik klassebegrepet brukes i programmering.

Dette illustrerer vanskeligheten med å bygge på studentenes forestillingsverden. Det at et ord kan brukes i så forskjellige betydninger, kan gjøre begrepsforståelsen hos studentene vanskelig. Der kan et forelesningsbasert opplegg virke klarere for studentene ved at begrepene kan beskrives klart, og i en rekkefølge som foreleser legger opp til. På den annen side vil foreleser ikke så lett oppfatte dersom

studentene assosierer klassebegrepet med for eksempel en skoleklasse. En foreleser har i en slik situasjon mindre mulighet for å bekrefte og korrigere studentenes eventuelle usikre begrepsoppfattelse. At det finnes et omforent begrepsapparat er viktig, siden begrepene i vårt opplegg skal brukes i en konstruktivistisk tankegang ved å bygge kunnskap om programmering slik det er omtalt i kapittel 6.6 Konstruktivistisk.

Det er risikabelt å basere seg på studentenes forestillingsverden i et undervisningsopplegg.
For det første kan metoden være vanskelig dersom det er mange studenter i gruppen, siden det er vanskelig å forutse de enkelte individenes bakgrunn og erfaring.

Jeg har tidligere beskrevet hvorledes vi i starten tok i bruk definisjoner som ikke var eksakte, for å prøve og tilpasse oss studentenes plattform når det gjaldt begreper. Eksempel på dette er Figur 17: Illustrasjon av klassebeskrivelser med objekter, i kapittel 9.1.1 Objekt-begrepet. Jeg skriver også at dette var et virkemiddel for å få studentene til å være klar over at det var forståelse av begrepene mer enn den eksakte definisjonen, som var vårt fokus. Figuren er også en illustrasjon på at vi av og til ikke var helt klar i vår språkbruk, idet vi i samme figur viste klassen og objektene som ble generert fra klassen. Jeg kan kritiseres for at jeg la opp til slik inkonsistens, relativt det som Unified Modeling Language (UML) notasjonen legger opp til. Jeg hevder at denne kritikken i det vesentlige er basert på at UML ikke blander klasser og objekter i samme skjema, men har klassediagram og objektdiagram.

En variant av det som er beskrevet ovenfor er når vi introduserte klassebegrepet (9.1.2 Klassebegrepet) med hjelp av først å ta i bruk begrepet mal. Situasjonsbetingelsen var her at vi i en overgangsfase hadde muligheten for å kunne si "…mal, eller klassebeskrivelse…". Det er verdt å merke seg at observasjonene viser at de også senere i kurset brukte begrepet mal istedenfor klasse. Dette kan indikere at studentene oppfattet begrepene som synonyme, eller at de i enkelte situasjoner foretrakk mal når de skulle konstruere sin kunnskap (6.6 Konstruktivistisk). Jeg kan kritiseres for at jeg innførte flere begreper enn det som er strengt tatt nødvendig for å definere objektorienterte begreper.

Mine observasjoner støtter entydig opp tilbakemeldingen fra studentene, om at det å bruke unøyaktige beskrivelser og erstatningsbegreper i en introduksjonsfase var greit og ikke gjorde dem forvirret. Dette kan komme av at det ikke var så mange begreper, og at studentene assosierte disse til en for dem kjent verden. Det er mer usikkert hvorledes dette hadde vært dersom mengden begreper og relasjonene mellom dem hadde vært mer kompleks.

Jeg hevder at situasjonsbetinget veiledning leder til en mer uformell kommunikasjon som kan gjøre det lettere for studenten å ta opp spørsmål som de lurer på. Et eksempel på dette er beskrevet i kapittel 9.1.2 Klasse-begrepet. Det er et spørsmål som ikke dreier seg om aktivitetene, men om hvorledes systemutviklerne av Karel J skriver koden. Siden dette er et spørsmål som ligger utenfor pensum, vil jeg hevde at det vitner om at studenten har en viss følelse av kommunikasjon hvor studentens interesser er sentral. Eller det kan være at studenten har en følelse av at

progresjonen er for svak og ønsker å påvirke denne. I diskusjonen om konstruktøren (9.1.3 Sub-klasse-begrepet) kan jeg registrere at vi faktisk måtte stoppe diskusjonen, og argumentet var da at de måtte bare godta at slik måtte konstruktøren være. Dette er et tilfelle hvor vi brøt med vårt opplegg basert på konstruktivistisk tankegang. Det er interessant og overraskende at de forstod at det var mekanismer i programmering som de på det tidspunkt burde akseptere. Studentene kan ha erfart at det kan være lurt å akseptere det en lærer sier, selv om studenten ikke har en klar forståelse av hvorfor det er slik (4.1 Mesterlære). Det kan også være at de ga uttrykk for at de forstod, fordi de ikke ville bruke mer tid på diskusjon, men komme videre med aktivitetene.

Mine observasjoner viser at situasjonsbetinget veiledning ikke alltid medfører umiddelbar læring. To eksempler på dette er: I eksemplet ovenfor angående diskusjonen om konstruktøren så ble malen akseptert, men læring som kunne overføres til andre situasjoner måtte komme senere i studiet. Det andre eksemplet illustrerer at studentene kan ha en motsatt oppfatning og er fra 9.1.4 Modulariseringsbegrepet, Aktivitet 4.2. Her sier en student "…kan ikke følge opp anmodinga din om…". Det er tydelig et bevisst valg om ikke å gjøre som veileder har sagt, og da med den begrunnelsen at det ikke hjelper. Nå ble de senere flinkere til å anvende modularisering. Begge eksemplene illustrerer at en instruktørs verbale beskrivelse nødvendigvis ikke medfører en umiddelbar læring hos studenten. Det er også en indikasjon på at studentene i vår setting føler en trygghet til å uttrykke sine egne synspunkter, og markere at de er kritiske til det som formidles. Det at de så klart uttrykker sine kritiske synspunkter kan komme fra vår bruk av situasjonstilpassing. Denne kan bidra til at studentene får en sterkere selvfølelse i læringssituasjonen.

Forskjellen på bruk av modularisering og gjenbruk mellom de to gruppene som ble videofilmet, kan brukes for å illustrere et aspekt av situasjonsbetinget veiledning. Den ene gruppen tok raskt i bruk mekanismen modularisering, mens den andre gruppen gjennom hele kurset hadde problem med å anvende teknikken. Her kunne vi tilpasse veiledningen ved at den gruppen som var rask med å ta i bruk modularisering kunne veiledes mer på teknikker for å effektivisere testing av moduler og gjenbruk, spørsmål som "Hva vil du at modulen skal gjøre med roboten?, Hvordan kan du lage en modul som gjør det?" I den andre gruppen hadde veiledningen mer fokus rundt relasjonen mellom objektbevegelse på skjerm og programkode, samt struktur. Her observerer jeg veiledning som "Hvorfor kolliderer roboten med trappen? Delen dere tester er litt stor, prøv å del opp i deler slik at dere kan teste hva som virker." Dette kommer etter at gruppen har prøvd å beskrive hva de observerer på skjermen og hva programmet skal gjøre. Elementene i forklaringen var uklar og gav inntrykk av at de var usikker på relasjon mellom bevegelse på skjerm og programkode. Min veiledning av gruppen skulle prøve å stimulere til at de kunne lære at enklere moduler kunne hjelpe dem i analysearbeidet og der igjennom bidra til at de lettere kunne produsere og gjenkjenne moduler. Selv om gruppen hadde litt utvikling i å anvende begrepene, så var det på slutten av kurset fortsatt stor forskjell på gruppene. Forskjellen kan komme av at gruppene var på forskjellig trinn i modningsprosessen. Det mest positive var at den svakeste gruppen erkjente at de nok burde bruke modularisering i større grad. Dette indikerer at de oppfattet poenget underveis, men burde hatt litt mer mulighet til å praktisere på oppgaver som hadde fokus på dette emnet.

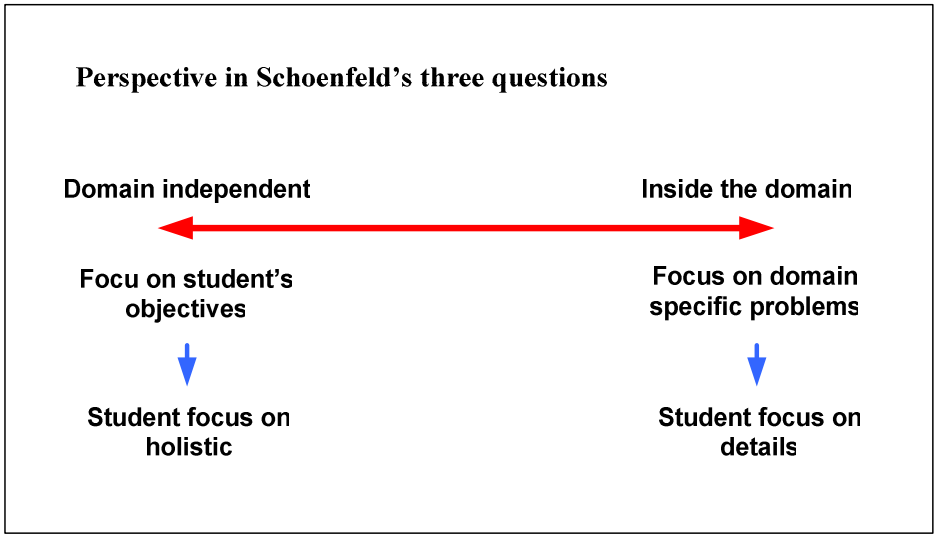
Et vesentlig bidrag for å kunne veilede grupper som har så forskjellig arbeidsmetodikk, er å tilpasse veiledningen til de betingelsene som presenteres gjennom studentenes verbale og skriftlige arbeider. Mine resultater viser også en klar indikasjon på at studentene kanskje har mye fokus på hint som de føler direkte bidrar til et positivt resultat på et aktuelt problem. Eksempel på dette er en gruppe som hadde et syntaksproblem: I en while setning tok de i bruk else og jeg fikk gruppen til å skrive opp syntaksstrukturen for if og while setninger. Da så de selv umiddelbart at de trengte en while og en if setning.

Bruk av situasjonsbetinget veiledning er en måte å inkorporere forelesninger og øvelser slik at læring blir mer et resultat av samarbeid (Herskin, 1994).

De tre spørsmålene

Som omtalt i kapittel 4.6 Veiledning baseres på tre spørsmål og 7.2.5 Veiledning, valgte jeg å basere veiledningen på Schoenfeld (1992), som bruker de tre spørsmålene: Hva gjør dere?, hvorfor gjør dere det?, og hvordan hjelper det dere?. I hvilken grad de tre spørsmålene kan ha påvirket resultatet, kan jeg bare si noe om, ut i fra hva studentene ga uttrykk for når vi spurte dem. Dette siden observasjonene i stor grad bare sier noe om begrepsforståelsen og lite om hvorledes denne ble bygget.

Det å formulere spørsmål fra domenet matematikk som Schoenfeld bruker, til vårt domene som er programmering av datamaskiner, kan være en utfordring. Som jeg har beskrevet i 4.6 Veiledning baseres på tre spørsmål, tolker jeg hans formuleringer til å rette seg mot at studenten skal uttrykke hva vedkommende ønsker å oppnå med med sitt forslag. Derimot har det tradisjonelt vært slik at når studenter skal lære objektorientert programmering, så har det vært naturlig at fokus hos undervisningspersonalet og studenten har vært mot fagspesifikke problem. For å illusterer problemstillingen vi som veiledere og studentene sto ovenfor, har jeg utviklet en modell jeg hevder viser sammenhengen mellom bruk av fokus i spørsmålene og hvilken studentreaksjon dette kan bidra til.



Figur 34: Perspektiv i Schoenfeld's tre spørsmål.

Figuren er en todimensjonal illustrasjon av perspektiv som kan anvendes ved Schoenfeld's tre spørsmål. Formen på spørsmålene kan i varierende grad tilpasses det aktuelle domenet. Dette er illustrert ved den røde linjen. Teksten under linjen er også knyttet til linjen og beskriver hvilken fokus teksten i spørsmålene kan ha. De blå pilene illusterer teksten: " Kan lede til at". Mens nederste del beskriver hva studenten fokuserer på.

Aristotles har i Metaphysics sagt at helheten er mer enn summen av alle deler som inngår (Aristotles). Dette er for meg en indikasjon på at det å flytte fokus fra detaljer til helhet kan bidra positivt til å bygge kunnskap i objektorientert programmering.

Jeg tolker spørsmålene til Schoenfeld slik at de i størst mulig grad skal formuleres slik at de ikke inneholder noen hint relatert til hva som kan være feil eller forslag til løsning på en eventuell feil. Basert på dette så ønsket jeg å bruke spørsmålene i størst mulig grad etter hans mal. Her må det sies at siden jeg ikke konkret hadde prøvd spørsmålene, måtte jeg være forberedt på at vi måtte situasjonstilpasse formen på spørsmålene til programmeringsdomenet dersom jeg følte at det var behov for det

For å illusterer noe av vanskeligheten med å formulere seg, bruker jeg en situasjon fra arbeidet med Aktivitet 3-2 (Vedlegg 3) som er omtalt i 8.1 Første dagen. Der var det en gruppe som erfarte at roboten etter å ha plukket opp første element, fortsatte rett frem inn i veggen. Årsaken til dette var at de ikke var bevisst at roboten startet med retning nord og etter første pickup stod med retning vest, slik at neste kall startet med et annet utgangspunkt. Mitt første spørsmål var "Hva gjør programmet?" Studentene startet da med en gjennomgang av koden hvor de fortalte hva de enkelte setningene gjorde. Etter en tid avbrøt jeg med: "Hva vil dere at objektet skal gjøre, og hvilke deler/moduler vil dere bruke for å få til det?" Dette gjorde jeg for å stimulere studentene til selv å klargjøre hvilke moduler de ville bruke og der igjennom avdekke eventuelle svakheter i overgangen mellom modulene. Eksempel på det jeg her omtaler som svakhet er at roboten forsatte rett frem mot veggen.

Min tanke med spørsmålet "Hva gjør programmet?" var at studentene skulle forklare hva programmet utførte. I det la jeg at de skulle forklare hva programmet fikk objektet på skjermen til å gjøre. Jeg kan se at spørsmålet inviterer studentene til å respondere med å beskrive koden i de enkelte settningene. Dette siden min setning retter seg mot programmet i motsetning til Schoenfelds form, som retter seg mot hva studenten gjør og ønsker å gjøre. Det er grunn til å tro at dette er et eksempel på at dersom studenten oppfatter at spørsmålet retter fokus mot syntaksen, så vil studentens fokus også være det (4.1 Mesterlære). Ser vi på formen jeg brukte etter at jeg avbrøt deres forklaring som var: "Hva vil dere at objektet skal gjøre, og hvilke deler/moduler vil dere bruke for å få til det?", ser vi at jeg er mer lojal ovenfor formen til de tre spørsmålene. Jeg må også erkjenne at studenten kan ha lettere for å oppfatte spørsmålet som mer spesifik og klarere relatert til det jeg ovenfor har beskrevet som min tanke med det.

Et annet eksempel på en formulering som jeg hevder er av en lignende form som den ovenfor er hentet fra 8.2 Andre dagen, Aktivitet 4-2. ”Hvorfor gjør dere dette akkurat sånn? Hvordan passer dette inn i resten av programmet?”. Studentene begynte å forklare hvordan de tenkte løsningen og oppdaget selv hva som var feil. Jeg hevder at formuleringene i spørsmålet bidro til at studentene i sin forklaring prøvde å begrunne og se sin egen kode i en helhet i løsningen fremfor å beskrive den enkelte kodelinje. Siden dette var en aktivitet som kom andre dagen, kan svaret også være mer basert på erfaring enn av den konkrete formulering i spørsmålet.

Hittil har jeg sett på min bruk av de tre spørsmål hvor jeg har henledet oppmerksomheten mot struktur og vurdere løsningen. Er det mulig å bruke spørsmålene rettet mot syntaks og valg av programmsetning? For å illustere dette bruker jeg flere eksempler fra vårt eksperiment. Jeg starter med et eksempel fra kapittel 8.3 Tredje dagen. En student sier ”vi prøver med en while setning” og så virker det ikke. Når jeg fikk studentene til å beskrive hva setningen skulle gjøre, og under diskusjonen, skrev jeg opp syntaksen til en if setning. Årsaken til dette var at testbetingelsen de brukte var beregnet for en if setning. Studentene foreslo selv at det var testbetingelsen som var problemet. De hadde ikke klart for seg forskjellen på testbetingelsen i while og if setninger.

Fra en annen situasjon: ”Hvorfor er testbetingelsen din slik?”. Her henleder veileder direkte mot testbetingelsen. Ved derimot å bruke de tre spørsmålene, kan en få studentene til å prøve og evaluere hva setningen skulle gjøre opp mot hensikten, og det gir større sjanse for at erfaringen kan bidra til generalisering av kunnskapen. Et annet eksempel: ”Hvorfor bruker du while?” Dette er en formulering som studenten lett oppfatter som et hint om å bruke if samt at den kan lede til at diskusjonene dreier seg om innholdet i programsetninger. En annen variant er at en veileder sier: ”Kan du bruke else i en while setning? Denne uttalelsen hjelper gruppen nærmere en løsning ved et klart hint om å bruke en if setning. Jeg kan forstå at en veileder lett ledes til å tilpasse spørsmålene til domenet slik jeg har beskrevet ovenfor. Hadde veileder brukt de tre spørsmålene, så måtte studenten prøve å beskrive hva han gjorde og begrunne det. De tre siste eksemplene bryter med noe av intensjonen med PBL metodikken (4.4 Problembasert læring) om at studenten skal tilegne seg kunnskap selv med lite veiledning. Følelse av mestring kunne nok ha blitt bedre om studentene selv fant frem til forbedringen av koden.

Selv om vi under planleggingen og under kurset var enig om at fokus i veiledningen skulle ligge på de tre spørsmålene, så må jeg innrømme at videoen dokumenterer at i praksis ble det litt annerledes. Spesielt er det i veiledningen under aktivitetene, at jeg noen ganger kan registrere at fokus, dreier fra det å veilede på begrepsforståelse ved hjelp av de tre spørsmålene, til fokus mot kompetanse på programsetningenes syntaks slik som i eksemplene ovenfor.

Som jeg har beskrevet tidligere er Schoenfelds formuleringer rettet mot hva studenten gjør, hvilken begrunnelse vedkommende har for det som er gjort og hvilken hjelp han har av det. Dette gjør at jeg hevder at spørsmålene kan betraktes som uavhengig av domenet. Med bakgrun i mine observasjoner hevder jeg at dersom spørsmålene i for sterk grad formuleres mot programmeringsdomenet, vil de i mindre grad bidra til at studenten fokuserer på det som jeg oppfatter som hensikten med spørsmålene. Nemelig, evaluere sitt eget arbeid opp mot problemet.

En årsak til at jeg hevder at det var lettere å kommunisere med studentene ved å bruke de tre spørsmålene, var fordi begge parter hadde en viss felles forståelse av hva veilederne kom til å kommentere (4.6 Veiledning baseres på tre spørsmål). Det å ha de tre spørsmålene, gjorde at jeg kunne fokusere på å vurdere deres løsningsforslag, å knytte dette til begrepsforståelsen som forslaget indikerte, eller som studentene gav muntlig uttrykk for. Dermed behøvde jeg ikke å hentyde noe konkret løsning.

Dersom jeg ikke hadde hatt de tre spørsmålene er det sannsynlig at jeg ville valgt en annen innfallsvinkel. Det kan være at jeg ville prøve å hjelpe litt ved å spørre om hvorfor roboten fortsetter inn i veggen, eller fått studentene til å legge merke til retningen på roboten i startposisjonen og retningen den hadde når den hadde plukket opp beeperen. Dette ville ha vært tips som kunne ledet gruppen mot et resultat som de med stor sannsynlighet ville ha oppfattet som vellykket. Det med bakgrunn i at tipsene direkte henleder på hvilke endringer som må gjøres i programkoden for å få et vellykket resultat på aktiviteten. Derimot tar de spørsmålene som jeg anvendte, utgangspunkt i at målet er å stimulere til at studentene selv evaluerer koden opp mot de objektorienterte begrepene som vi brukte.

Jeg erfarte at de tre spørsmålene bidro til at det ble lettere å holde fokus på begrepene i veiledningen. For å illustrere dette bruker jeg en situasjon som jeg har omtalt tidligere i kapittelet med spørsmålet”Hva gjør programmet” (8.1 Første dagen, Aktivitet 4-2). Mitt neste spørsmål var ”Hva vil dere at objektet skal gjøre, og hvilke deler/moduler vil dere bruke for å få til det?”. Jeg hevder at formen på de tre spørsmålene gjør det lettere å knytte disse opp mot de begrepene vi ønsket å fokusere.

Våre observasjoner viser også at denne måten å veilede på, kan bidra til at studentene opplever at begrepene er viktigere enn for eksempel syntaksen. Dette siden veilederne retter fokus på begreper, også når studentene kommer med spørsmål som er relatert til syntaks. Når instruktøren i sin veiledning fokuserer på begreper, vil det være naturlig at studenten gjør det samme (4.1 Mesterlære). Vi spurte studentene om vårt fokus på begreper hjalp dem å fokusere på begrepene, og en illustrasjon på svarene er: ”Når vi skal spørre så må jo vi være forberedt”, og ”Når dere har snakket med oss er det lettere å fortsette å prate om de samme tingene.” Disse tilbakemeldingene støttet opp om våre observasjoner. Funnene er også i overenstemmelse med Schoenfeld (1992). Studentene opplevde at begrepsforståelse var viktigere enn syntaks for å lage løsninger. Eller som en sa det: ”Feil i setningen er lett å finne i kompilatoren, men for å bruke objektene må du skjønne det”.

Da vi hadde en oppsummering, kommenterte studentene at spørsmålene hjalp dem til å tenke selv og som en sa: ”…tvinger oss til å overføre kunnskap fra en situasjon før, til det vi holder på med”. I dette og lignende uttalelser legger jeg at de måtte vurdere tidligere utførte aktiviteter, og prøve å finne noe som kunne overføres til den aktuelle situasjonen. Dette ”noe” ser jeg av observasjonene også inneholder element av klipp og lim, samt begrepsanvendelsen.

Hittil har jeg omtalt komunikasjonen mellom veileder og student. Jeg skal prøve å se litt nærmere på kommunikasjonen mellom studentene. Som jeg har beskrevet tidligere uttrykte studentene at de brukte de tre spørsmålene internt i gruppen under arbeidet. Et eksempel på dette er utalelsen ovenfor: ” Når dere har snakket med oss er det lettere å fortsette og prate om de samme tingene”. Våre observasjoner og videoopptakene bekrefter dette. I notatene fra arbeidet med Aktivitet 3-2 (Vedlegg 3) har jeg notert en episode: ”Hva gjør egentlig roboten i den modulen?” Her spør den ene studenten den andre i gruppen om å få en utdyping av det roboten gjør, istedenfor å be om en beskrivelse av koden. Et annet eksempel fra opptakene av Aktivitet 6 (Vedlegg 3): ”Hvorfor bruker du modulen fra ur_robot, istedenfor fra ur_robot2?” Her peker studenten på skjermbildet hvor koden vises. En annen fra samme aktivitet: ”Kan den klassen hjelpe oss?” Kommentaren kom mens en gruppe lette etter klassedefinisjoner som de hadde brukt tidligere i kurset.

Det siste eksemplet er også en illustrasjon på at studentene særlig anvendte de tre spørsmålene der studentene opplevde at de hadde litt alvorlige problemer, som for eksempel, mistanke om at de var på feil spor. Dette var jo nettopp et av målene med å anvende de tre spørsmålene, at studentene skulle motiveres til å være kritisk til sine egne forslag. Det at spørsmålene fungerte så bra, kan komme av at de bygger opp om PBL metodikken. Det skjer ved at spørsmålene leder studentene til å lete frem nødvendig informasjon og vurdere denne opp mot problemet de jobber med. Formen gjør det også mer naturlig å knytte spørsmålene opp mot begreper slik det er gjort i Aktivitet 3-2 ovenfor, ved for eksempel å anvende det tredje spørsmålet: ”Hva vil dere at objektet skal gjøre?” Legg her merke til at mitt spørsmål bruker andre ord enn Schoenfeld, men det har samme formulering og innhold. Denne forskjellen ligger i at Schoenfeld sin tekst har en generell form, mens min formulering er knyttet opp mot en spesiell situasjon.

Oppsummerer jeg mine funn angående bruk av de tre spørsmål, er det for det første at dersom perspektivet i spørsmål flyttes fra studentens argumentasjon og hensikt med programkoden til selve programkodens utforming, så er det vanskelig å få de tre spørsmål til å fungere etter Schoenfelds hensikt. Hensikten har jeg beskrevet som å få studenten til å vurdere sine løsningsforslag opp mot problemet og være villig til å forkaste løsninger som ikke ser ut til å fungere relativt målet. Det er nok ikke uten grunn at Schoenfeld har gitt spørsmålene et perspektiv som er generisk relativt domenet de brukes i.

9.3.2 Progresjon

Det andre hovedtema i denne oppgaven er studentdrevet progresjon.

I denne oppgaven er opplegget for å få til studentdrevet progresjonen nært knyttet til situasjonsbestemt veiledning slik denne er beskrevet i forgående kapittel. I mitt materiale har jeg vanskelig for å skille på i hvilken grad situasjonsbestemt veiledning eller studentdrevet progresjon har bidratt til resultatet. Jeg kan bare si noe om elementenes påvirkning, men resultatet betrakter jeg som i det vesentlige komplimentært. Et eksperimentopplegg for å prøve å avklare de enkelte elements betydning kan være av interesse å se på i et fremtidig prosjekt.

Når vi i vårt eksperiment ville prøve ut studentdrevet progresjon, ønsket vi å bruke en del virkemidler. Her var det sentralt at den enkelte students motivasjon skulle baseres på mestringsfølelse (4.2 Mestre) og kontinuerlig følelse av læring. I det siste ligger det at vi ønsket at studentene under hver aktivitet skulle ha en følelse av at de lærte noe som de opplevde som nyttig for å løse et problem (4.4 Problembasert læring).

Jeg har ingen mulighet til å kunne uttale meg om studentenes egen opplevelse. Jeg kan bare basere meg på min tolking av mine observasjoner og hva studentene selv ga uttrykk for i samtalene. Når det gjaldt mestringsfølelse som virkemiddel i studentdrevet progresjon så viser dataene klare fellestrekk. En uttalelse som godt representerer dette er: ”Det er viktig å få det til, og skjønne hva man gjør” denne uttalelsen tolker jeg som et uttrykk for at studentene utviklet forståelse for at det er ikke nok å få det til, du må skjønne hva du gjør og hva det kan brukes til. Utsagnet ble fulgt opp av en annen student med: ”Det er mer moro når jeg forstår åssen hjelp jeg får av det”. Disse uttalelsene tolker jeg som at studentene uttrykker at de mestrer aktivitetene og oppfatter de som motiverende (4.2.3 Motivasjonsfaktor). Jeg hevder at motivasjonen til den enkelte student er vesentlig når læring skal baseres på studentdrevet progresjon.

Uttalelsene ovenfor kan også knyttes til vårt PBL-baserte opplegg slik det er beskrevet i kapittel 6 Begrunnelse for eksperimentet og 7 Planlegging av eksperimentet. Jeg hevder at det for studentene var viktig å erfare at kunnskap om de begrepene som vi tok opp, kunne bidra til å løse et aktuelt problem som de jobbet med. Da kan det være spørsmål om i hvilken grad det var mestringsfølelse eller vårt PBL-opplegg som bidro.

Det at vi lot studentenes utvikling av kunnskap være førende for progresjonen i kurset, var avhengig av at vi klarte å registrere kunnskapsnivået til individene i gruppen ved de sjekkpunkt vi la opp til. Typiske sjekkpunkt var basert på samtalene før og etter hver aktivitet. Men vi trakk også inn erfaringene vi gjorde når vi veiledet på den enkelte aktivitet. Resultatene mine relatert til bruk av disse sjekkpunktene, viser at dette opplegget fungerte godt og studentene etter kurset gav en meget positiv tilbakemelding på deres opplevelse av dette opplegget. Den eneste bemerkningen var knyttet til aktiviteten med Robocode, som er beskrevet i kapittel 8.2 Andre dagen. Bemerkningen var at spranget fra aktivitetene de hadde jobbet med før var for stor. Knyttet til progresjon kan jeg tolke dette som at vår sjekkpunktkontroll ikke var tett nok eller av tilstrekkelig kvalitet til å avsløre dette i forkant. I ettertid vil jeg hevde problemene også avdekker at studentdrevet progresjon, ikke er tilstrekkelig for at studentene har kunnskap og erfaring nok til å anvende sin kunnskap i mer komplekse oppgaver. Dette kan være en indikasjon på at når det anvendes studentdrevet progresjon vil det være en fordel at progresjonskrittene er tilstrekkelig små for at opplegget skal fungere etter sin hensikt. Et annet alternativ kan være å bruke repetisjoner for å gi studentene tilstrekkelig erfaring.

Andre moment som kan ha påvirket progresjonen er at studentene hadde mye hjelp av at de jobbet i par og kunne supplere hverandres kunnskap. Den uformelle

diskusjonen i plenum gav også mulighet for å få kunnskap om andre innfallsvinkler og hvorledes andre tenkte på de samme problemstillingene de selv jobbet med.

Vi introduserte begrepsbeskrivelser ved hjelp av definisjoner som var mindre presise og erstatningsbegreper som studentene kjente fra sin egen forestillingsverden som fundament (9.3.1 Situasjonsbestemt veiledning og de tre spørsmålene). Dette kan ha bidratt til at studentene raskere kunne komme i gang med hver enkelt aktivitet.

Mine observasjoner og studentenes uttalelser bekrefter at i en introduksjonsfase var slik bruk av begreper greit, og ikke gjorde de forvirret. Dette kan komme av at det ikke var så mange begreper og at studentene assosierte disse til en for dem kjent verden. Det er mer usikkert hvorledes dette hadde vært dersom mengden begreper og relasjonene mellom dem hadde vært mer komplekse.

Vedkommende som forestår kursopplegget, bør ha evne til å tilpasse og lede progresjonen i gruppen. Hva gjør en for eksempel dersom en del av gruppen gir uttrykk for at de ikke forstår noen av assosiasjonene som brukes? Skal en da satse på å forklare at det forstår de senere, eller skal en stoppe opp og sikre seg at en får med alle? Dette kan jo virke forstyrrende på de som føler at de ikke har noe problem med forståelsen. Siden hele tankegangen i objektorientert programmering bygger på innsikt i begrepers anvendelse, var vi innstilt på å få med alle før vi fortsatte.

Den eneste gangen jeg har registrert at vi brøt med dette prinsippet er omtalt i kapittel 9.3.1 Situasjonsbetinget og de tre spørsmålene, i eksemplet med konstruktøren. Dette er et eksempel på at bruk av studentdrevet progresjon kan fungere selv om det er elementer som studenten ikke nødvendigvis forstår programmeringsmekanismen i, men aksepterer at det bør være slik.

Mine data indikerer at studentdrevet progresjon er egnet også når gruppene har forskjellig arbeidsmetodikk slik dette er beskrevet i 9.3.1 Situasjonsbetinget og de tre spørsmål. Med arbeidsmetodikk menes her at gruppenes praktisering av begreper som modularisering og gjenbruk var klart forskjellig, men dette var ikke til hinder for progresjonen med aktivitetene. Jeg vil hevde at dette er begreper som er avhengig av studentens modenhet i faget. Dersom dette er korrekt så indikerer mine data at studentdrevet progresjon fungerer ved læring av denne type begreper når studentene har forskjellig modenhet i faget.

Mine resultater indikerer at studentdrevet progresjon vil fungere i et kurs som brukes for å introdusere objektorientert programmering. Svakheten i resultatene er ekstern validitet (3.7 Innsamling av data) med bakgrunn i antall deltakere og antall dager på vårt kurs. Med bakgrunn i mine resultater kan det være interessant å prøve temaet i et kurs som går over et semester.

9.4 Noen erfaringer

9.4.1 Notat versus video

Vi hadde i vårt opplegg basert oss på inntil 25 personer i gruppen, nå fikk vi jo betydelig færre. Dette medførte at vi fikk en god oversikt over det som ble gjort av arbeid under aktivitetene. Det ble god tid til å notere ned observasjoner underveis. Jeg har registrert at notatene har en annen karakter enn videoopptakene. De er meget kortfattet og fokuserer på det jeg syntes var viktig der og da. Mens videoopptakene kan gi mulighet for å vurdere en situasjon i flere perspektiv. ”En skal være oppmerksom på at under tolkning av videoopptak er observatørens erfaring og bakgrunn en påvirkning på perspektivet som kommer til anvendelse under analysen.” (Kvale, 2001). I dette ligger det at jeg ikke kan oppfatte mine observasjoner som en sannhet, men anvende en kritisk evaluering av om det er andre konklusjoner som kan komme til anvendelse.

Jeg har sammenlignet mine notater gjort under eksperimentet, med det jeg kan observere fra videoopptakene. Overensstemmelse mellom de to kildene er litt overraskende. Jeg erfarte at det er stor overensstemmelse når det gjelder flere av konklusjonene fra mine notater og hva som ble observert på videoen. Forskjellen ligger i at via videoen så har jeg data som jeg kan spole tilbake i, og dermed er det lettere å kvalitetssikre observasjonene.

Men jeg har også observert at det er noen av mine notater som ikke helt stemmer med det jeg kan observere av videoen. Et eksempel på dette er at jeg noterte at: ”1H har problem med å plassere if, i while, programmet går for sakte”. Dette betyr at gruppen fremst på høyre side plasserer if setningen inne i while setningen og derfor kommer testen hver gang de utfører løkken. På videoen kommer det klart frem at studentene har sett dette problemet, og faktisk tester ut om det går saktere når if setningen er inne i løkken, enn med if setningen på utsiden av løkken. Grunnen til at jeg ikke observerte det, kan være at studentene kanskje foretrekker å prøve ut slike ting når en instruktør ikke observerer dem. Det kan være en redsel for å dumme seg ut ovenfor en instruktør, eller rett og slett at de lettere finner tid til å eksperimentere når de føler at ingen overvåker dem.

9.4.2 Robocode

At vi valgte dette systemet for å ha det som ”rosin i pølsa” og som noe å strekke seg etter, var ut i fra våre mål om at studentene skulle erfare gjennom aktiviteter. Det var også fordi vi følte oss sikre på at de aller fleste hadde erfaring fra spill på datamaskin. Vi håpet at anvendelsen skulle bidra til at de avsluttet kurset med det som Albert Bandura (1986 og 1998) i sin beskrivelse av kognitiv teori beskriver som en følelse av mestring. Dette er tidligere omtalt i kapittel 4.2 Mestre.

Vi håpet at overgangen fra programmene i KarelJ til programmene i Robocode skulle oppleves av studentene som et lite steg i kompleksitet (7.3.1 Valg av verktøy). Men det viste seg at de fikk vanskeligheter i den nye settingen.

Dette kan komme av flere årsaker. En er at KarelJs verden er enklere, idet den er statisk med vegger og beepere. Mens Robocode har en dynamisk verden hvor roboten skal forholde seg til andre roboter som er i bevegelse. Dette bidrar til at operasjonene til KarelJ gir mye enklere funksjonalitet enn de som finnes i Robocode (Vedlegg 5). Dette betyr også at algoritmene var av en annen karakter enn studentene var vant med fra tidligere aktiviteter. En annen årsak kan være at det kreves en god del mer programmering for å få Robocode til å utføre operasjoner på skjermen, sammenlignet med hva som kreves for KarelJ. Dette bidrar til at den mengden programmering som studentene måtte gjøre for å bruke Robocode, var såpass mye større og kompleks at de fikk problemer.

Det viste seg at studentene fikk vanskeligheter med å anvende begreper som de hadde jobbet med i de tidligere aktivitetene. Dette var begreper som for eksempel objekt, sub-klasse, moduler, gjenbruk og programsetninger som if og while. Vanskelighetene er et eksempel på at det å konstruere kunnskap må bygge på kunnskap som studenten besitter fra før (6.6 Konstruktivistisk). Dersom studenten opplever at spranget i kunnskapsnivå mellom grunnlaget og det aktuelle læringsmål blir for stort, vil det være vanskelig å konstruere ny kunnskap. Nå vil jeg hevde at det ikke var stort sprang i kunnskapsnivået knyttet til anvendelse av de enkelte begreper, men kompleksiteten og mengden av kode var uvant og vi fikk derav et sprang i erfaringsgrunnlag mellom aktivitetene i KarelJ og Robocode.

I ettertid innser jeg at det som kanskje bidro sterkest til at studentene slet mer enn vi hadde påregnet, var at de var avhengig av å se på koden til en tanks som var i systemet fra før, for der å finne moduler som de kunne anvende på sin egen tanks. Årsaken til at de fikk problemer, ligger i to elementer. Det første er at koden var så stor og komplisert at de hadde vanskeligheter med å orientere seg, samt skaffe seg en oppfatning av hva koden egentlig utførte av operasjoner. Det andre er at de på grunn av tempoet som tanksen beveget seg med, hadde vanskelig med å knytte sammen bevegelsen de kunne se på skjermen mot en aktuell modul i koden.

Så erfaringen blir at vi burde hatt et lengre kurs, slik at vi kunne introdusert større og mer komplisert kode i KarelJ før vi prøvde med Robocode.

10 Konklusjon

I dette kapitlet ser jeg først på de vesentlige didaktiske prinsipp (9.2 Betydningen av didaktiske elementer) som kom til anvendelse ved kurset (8 Gjennomføringen av eksperimentet). Så tar jeg en gjennomgang av de objektorientertebegreper jeg så nærmere på (9.1 Drøfting av læring av OOP begreper), før jeg konkluderer kursopplegget. I 10.1 Videre arbeid, tar jeg opp en del temaer som det kan være av interesse å studere nærmere.

Jeg tar utgangspunkt i min hypotese slik det er beskrevet i 1.2 Problemstilling. ”Prøve ut om en didaktikk som baseres på veiledning og studentdrevet progresjon, kan bidra til begrepsbygging av objektorienterte begreper”. Hypotesen ble brukt til å utvikle en didaktikk basert på at studenter skulle konstruere kunnskap (6.6 Konstruktivistisk) i et introduksjonskurs om objektorientert programmering. I dette ligger det at didaktikken ikke skal støtte opp under tradisjonelle forelesninger og arbeid med øvelser, men at situasjonsbetinget veiledning og studentdrevet progresjon skal være sentral for å bygge en forståelse for programmering. For at dette skal skje må veiledningen brukes for å innføre begrepsapparat og gi studenten forståelse av struktur og sammenhenger (1.2.1 Læresituasjonen).

For å motivere studentene tok vårt didaktiske opplegg utgangspunkt i at studentene skulle få en følelse av mestring (4.2 Mestre). Får å få til dette brukte vi flere elementer. Jeg kan ikke si noe om i hvilken grad det enkelte element gav bidrag til resultatet. Men mine vurderinger tar utgangspunkt i at resultatet er komplimentært.

To element som ble brukt var problembasert læring (4.4 Problembasert læring) kombinert med parprogrammering (4.5 Parprogrammering). Studentenes uttalelse og mine observasjoner gir klar indikasjon på at det å bruke disse to elementene var god hjelp for arbeidet med begrepsbyggingen. Dette fordi det ledet studentene til måtte jobbe aktivt og la til rette for å kunne utfylle hverandre med hensyn på erfaring og kunnskap (9.2.1 PBL og parprogrammering). Det viste seg også at disse to metodene er godt egnet å kombinere med at det skriftlige materialet vi delte ut var meget kort (7.2.2 Skriftlig materiale) og et grafisk brukergrensesnitt (Borge, 2004).

Dataene viser også at vi i vårt valg med å legge opp til at studentene skulle få mange begrepsvarianter å assosiere til, i enkelte tilfeller ikke bidro til å avklare studentenes begrepsoppfattelse.

Når det gjelder selve veiledningen valgte jeg to strategier. Den første var at jeg fokuserte på situasjonsbetinget veiledning (2.7 Veiledning) ved hjelp av muntlig kommunikasjon med litt støtte av figurer og kort tekst. Den andre var at jeg valgte å prøve veiledning basert på de tre spørsmålene: Hva gjør dere?, hvorfor gjør dere det? og hva skal dere bruke det til? (Schoenfeld, 1992). Legg merke til at formen rettes mot at studenten skal evaluere hva som gjøres og hva vedkommende ønsker å gjøre. En god illustrasjon er hva en student sa om de tre spørsmål: ”..de tvinger oss

til å tenke selv." Begge veiledningsstrategier fungerte godt med hensyn på å lære de objektorienterte begreper.

Mine resultater indikerer at studentdrevet progresjon vil fungere i et kurs som brukes for å introdusere objektorientert programmering. Men det er ikke tilstrekkelig for å sikre at studentene har kunnskap og erfaring nok til å anvende sin kunnskap i mer komplekse oppgaver. Det vil være en fordel at progresjonskrittene er tilstrekkelig små eller at det brukes repetisjoner for at opplegget skal fungere etter sin hensikt.

Når det gjelder læring av de objektorienterte begrepene: Objekt, klasse, subklasse, modularisering, og gjenbruk avdekket observasjonene at resultatene ikke var uniforme, og jeg går derfor gjennom de enkelte begreper.

Objektbegrepet (9.1.1 Objekt-begrepet) var det første begrepet vi tok opp og observasjonene viser at studentene nokså umiddelbart var i stand til å forstå hvorledes begrepet anvendes i programmering. Det var også et begrep som studentene hadde en klar og konsekvent anvendelse av under kurset. Innføring av klasse (9.1.2 Klasse-begrepet) og subklassebegrepene (9.1.3 Sub-klasse-begrepet) ble gjort via begrepet mal og dette hevder jeg var vellykket. Et problem studentene hadde med disse to begrepene var at de glemte hvilken klasse som hadde hvilke egenskaper. Den primære årsaken til dette ligger sannsynligvis i at vi brukte klassebetegnelser som lignet for meget på hverandre, for eksempel Robot, Ur_Robot og Ur_Robot2. Når studentene i tillegg laget egne subklasser som bygget på disse og med navn som lett kunne forveksles, er det ikke å undres over at dette av og til kunne være et problem. Derfor burde opplegget vårt ha hatt en klarere struktur med hensyn på objektnavn og klassenavn.

Begrepene modularisering (9.1.4 Modularisering-begrepet) og gjenbruk (9.1.5 Gjenbruk-begrepet) klarte studentene raskt å beskrive og anvende i løsning, men å abstrahere og generalisere over i andre situasjoner var et større problem enn jeg hadde forutsatt. Resultatet indikerer også at opplegget burde ha bygget erfaring i flere trinn, samt at disse to begrepene representerer begreper som krever mer erfaring før studentene behersker anvendelse i programmering.

Den interne validitet vil jeg hevde er god, med bakgrunn i vårt kildemateriale. Mens den eksterne validitet er betydelig mer usikker, spesielt med bakgrunn i at det var få studenter som deltok i eksperimentet. Vi burde ha planlagt eksperimentet slik at vi hadde fått deltagere som bedre kunne representere populasjonen som er aktuell for introduksjonskurs i programmering (3.7 Innsamling av data).

Resultatet av eksperimentet kan oppsummeres til at det var vellykket å innføre de objektorienterte begrepene jeg ser på i denne rapporten, med hjelp av vårt didaktiske opplegg.

Men siden jeg undervurderte studentenes behov for repetisjoner (9 Drøfting og funn) vil jeg i et ordinært kurs anbefale at det gjøres endringer som gir rom for dette.

10.1 Videre arbeid

Dette kapitlet tar for seg en del temaer som det kan være av interesse å se nærmere på, og da gjerne med mulighet for å sammenligne data fra undersøkelser med alternative metoder for gjennomføring av kurs.

Det kan være av interesse å sammenligne et slikt intensivkurs kontra et som har et opplegg basert på færre timer per dag. Det vil si å bruke flere dager og da gjerne med opphold mellom dagene slik at studentene kan jobbe litt på egen hånd. En annen variant kan være å ta i bruk flere repetisjoner for å hjelpe studentene å bygge kunnskap. Dette siste gjelder særlig for å lære begreper av typen modularisering og gjenbruk. Siden studentene hadde problem med anvende disse teknikkene (9.1.4 Modularisering-begrepet og 9.1.5 Gjenbruk-begrepet) i andre problemstillinger, vil det også være interessant å prøve ut andre oppgavetyper som vektlegger disse teknikkene sterkere.

Det vil for eksempel være av interesse å få en grundigere vurdering av vårt didaktiske opplegg med særlig fokus på å teste ut mot mer kompliserte programkonstruksjoner. Et eksempel på dette er mer komplekse sub-klasse konstruksjoner. Min drøftning av vår bruk av de tre spørsmål (Schoenfeld, 1992) avdekker problem med tilpassing til domenet. Å få prøvd ut i hvilken grad det går å tilpasse spørsmålene til domenet: Å lære at programmering, kan være interesant å studere i et semesterkurs.

I kapittel 9.2.1 PBL og parprogrammering står det at jeg har vanskeligheter med å klargjøre i hvilken grad hver av metodene bidro til resultatet, eller om det er et produkt av komplimentær bruk. Dette kan være interessant å se nærmere på når det gjelder introduksjonskurs i programmering.

Det vil være interessant å se nærmere på om studentenes kunnskap og erfaringer i de forskjellige tiår endrer forutsetninger for å kunne lære etter forskjellige metoder. For eksempel om studentene før hadde bedre forutsetninger for å tilegne seg kunnskap etter metoder basert på overføring av kunnskap fra lærer til student enn dagens studenter har?

Siden mine resultater har en svak ekstern validitet, kan det være ønskelig å oppskalere vårt kurs til et prosjekt som kan bidra med en sterkere ekstern validitet. Dette kan gjøres ved at et introduksjonskurs i programmering kjøres parallelt med to opplegg: Et tradisjonelt forelesningsbasert opplegg og et opplegg basert på våre resultater.

Referanser

Aristotles
http://en.wikipedia.org/wiki/Holistic#Holism_in_science

Bandura, Albert; (1986)
Social foundations of thought and action: Asocial cognitive theory.
ISBN**:** 013815614X
Prentice Hall

Bandura, Albert; (1998)
Encyclopedia of mental health, (Red: Fredman, H.)
San Diego: Academic press

Barg, Mike; Fekete, Alan; Greening, Tony; Hollands, Owen; Kay, Judy; Kingston, Jefferey H.; Crawford, Kathryn (2000)
Problem-Based Learning for Foundation Computer Science Courses
University of Sidney
Computer Science Education 10(2), 2000, pp1-20.
http://www.cs.usyd.edu.au/~judy/Teach/cse_pbl99.pdf

Barnes, David, J. og Kölling, Michael; (2005)
Objects First With Java, A Practical Introduction using BlueJ
ISBN 0-13-044929-6
Prentice Hall

Baskerville, Richard L. og Wood-Harper, A. Trevor; 2002
Myers, Michael D. og Avison, David; (red, 2002)
Qualitative Research in Information Systems
ISBN 0 7919 66323
SAGE Publications Inc.
2455 Teller Road ,Thousand Oaks, California 91320

Bergin, Joseph; (2000)
http://csis.pace.edu/~bergin/papers/Whynotproceduralfirst.html
Pace Univeristy

Bereiter, C., & Scardamalia, M. (in press, 2005 okt.).
Learning to work creatively with knowledge.
In E. De Corte, L. Verschaffel, N. Entwistle, & J. van Merriënboer (Eds.),
Unravelling basic components and dimensions of powerful learning environments.
EARLI Advances in Learning and Instruction Series.
http://ikit.org/fulltext/inresslearning.pdf

Bishop, Judy M.; (1998)
Java Gently
ISBN 0-201-34297-9
Addison Wesley Longman Limited

Edinburgh Gate, Harlow, Essex CM20 2JE, England
Borge, Richard Edvin; (2004)
Teaching OOP using graphical programming environments, An experimental study
University of Oslo, Department of Informatics

Burch, Kurt.; (2001)
The Power of Problem-Based Learning: A Practical "How To" for Teaching Undergraduate Courses in Any Discipline
Barbara J. Duch, Groh, Susan E. og Allen, Deborah E. (red)
ISBN**:** 1579220371
Stylus publishing

Campbell, Donald T. og Stanley, Julian C.; (1963)
Experimental and Quasi-Experimental Designs for Research
ISBN: 0395307872
Houghton Mifflin Company

Chase J. D. og Okie E. G.; (2000)
Combining Cooperative Learning And Peer Instruction In Introductory Computer Science.
ACM SIGCSE 32,1, 372-376

COOL; (2006)
Comprehensive Object-Oriented Learning: The Learner's Perspective
Annita Fjuk, Amela Karahasanovic og Jens Kaasbøll (red)
ISBN: 83-922337-4-3
Informing Science Press

COOL: Resturang eksemplet
http://www.intermedia.uio.no/cool/complex.html

COOL: Prosjektbeskrivelse
http:/heim.ifi.uio.no/~kristen/FORSKNINGSDOK_MAPPE/F_COOL1.html

COOL: Prosjekt plan
http://www.intermedia.uio.no/cool/docs/projectplanmai2003.pdf

Craig A.; (1998)
Peer Mentoring Female Computing Students-Does it Make a Difference?
Proceedings of the Third Australasian Conference on Computer Science Education,
ACM, 1, 41-47

Glanz, K.; Rimer, B.K. og Lewis, F.M.; (2002).
Health Behavior and Health Education. Theory, Research and Practice.
ISBN: 0787957151
Jossey-Bass; 3 edition

Gonzalez, Rafael C. og Woods, Richard E.; (2002)
Digital Image Processing
ISBN 0201180758
Prentice Hall, Pearson Education

Greening, Tony; Kay Judy og Kingston, Jeff (1997)
Departmental evaluation of the PBL trail – semester 1 1996
Basser Department of Computer Science
University of Sidney
http://www.cs.usyd.edu.au/~judy/PBL/report_dept.html

Grønmo, Sigmund; (2004)
Samfunnsvitenskapelige metoder
ISBN 82-7674-224-6
Fagbokforlaget

Halvorsen, Knut; (2003)
Å forske på samfunnet, 4. utgave
ISBN 82-02-22654-6
Cappelens Akademisk Forlag

Hanks, Brain; (2005)
Student Performance in CS1 with Distributed Pair Programming
ITiCSE'05, June 27-29, 2005, Monte de Caparica, Portugal.
10th annual SIGCSE Conference on
Innovation and technology in computer science education.

Harvey, Linda J. og Myers, Michael D.; 2002
Myers, Michael D. og Avison, David; (red, 2002)
Qualitative Research in Information Systems
ISBN 0 7919 66323
SAGE Publications Inc.
2455 Teller Road ,Thousand Oaks, California 91320

Herskin, Bjarne; (1994)
"Kap2: Alternativ Hands On-undervisnig" Brukervenlig EDB undervisning,
ISBN: 87-571-1720-9
Teknisk Forlag, København.

Herzog, Christian Johan; (2005)
Pair programming and learning, Cand. Scient oppgave,
University of Oslo, Department of Informatics

Hofset, Arnold; (1995)
Å undervise studenter
ISBN 82-00-21526-1
Universitetsforlaget AS, postboks 2959, 0608 Oslo

INF1000: Grunnkurs i objektorientert programmering
http://www.uio.no/studier/emner/matnat/ifi/INF1000/

INF1000: Undervisningsplan
http://www.uio.no/studier/emner/matnat/ifi/INF1000/h03/undervisningsplan.xml

Jacobsen, Dag Ingvar;(2000)
Hvordan gjennomføre undersøkelser? Innføring i samfunnsvitenskapelig metode
ISBN 82-7634-292-2
Høyskoleforlaget

Jensen, Karen; (1999)
Nielsen, Klaus og Kvale, Steinar (red)
Mesterlære, læring som sosial praksis
ISBN 82-417-1011-9
Ad Notam Gyldendal

Kaasbøll, Jens; (2002)
INF-DID informatics
Department of Informatics, University of Oslo
http://www.ifi.uio.no/infdid/pensum.shtml

Kvale, Steinar; (2001)
Det kvalitative forskningsintervju
orginal titel : InterViews – An introduction to Qualitative Research Interviewing, utgitt av Sage Publications i 1996.
ISBN 82-417-0807-6
Gyldendal Akademisk

Kölling, Michael; (2005)
Position summary
Resolved: Objects First has failed
ISSN: 0097-8418, 2005
SIGCSE 2005

Lattanzi, Mark og Sallie Henry, Sallie; (1996)
Teaching the Object-oriented Paradigm and Software Reuse
Computer Science Education, V7, N1

Laurillard, Diana; (1993)
Rethinking University Teaching: A framework for the effective use of educational technology
ISBN: 0415092892
Routledge, London and New York.

Lieux, Elisabeth M.; (2001)
The Power of Problem-Based Learning: A Practical "How To" for Teaching Undergraduate Courses in Any Discipline
Barbara J. Duch, Groh, Susan E. og Allen, Deborah E. (red)
ISBN**:** 1579220371
Stylus publishing

Luger, George F.; (2005)
Artificial Intelligence, Structures and Strategies for complex Problem Solving
ISBN 0 321 26318 9
Pearson Education Limited

McDowell C.; Werner L.; Bullock H. og Fernald J.; (2002)
The Effects of Pair-Programming on Performance in an Introductory Programming Course.
SIGCSE Bulletin 34, 1, 38-42

Mughal, Khalid Azim; Hamre, Torill og Rasmussen, Rolf W.; (2003)
JAVA som første programmeringsspråk
ISBN 82-02-23274-0
Cappelen Akademisk Forlag

Myers, Michael D. og Avison, David; (red, 2002)
Qualitative Research in Information Systems
ISBN 0 7919 66323
SAGE Publications Inc.
2455 Teller Road ,Thousand Oaks, California 91320

Nielsen, Klaus og Kvale, Steinar (red, 1999)
Mesterlære, læring som sosial praksis
ISBN 82-417-1011-9
Ad Notam Gyldendal

Orlikowski, Wanda J. og Baroudi, Jack J.; (1991)
Studying Information Technology in Organizations: Research Approaches and Assumptions
Information System Research, 2, 1

Ormrod, Jeanne Ellis; (2004)
Human learning, 4th edition
ISBN 0-13-094199-9
Pearson Prentice Hall

Pajares, Frank og Schunk, Dale H.; (2001)
Self-beliefs and school success: Self-efficacy, self-concept, and school achievement. Chapter in R.Riding & S. Rayner (Eds.), (2001) Perception (pp. 239-266)).
London: Ablex Publishing.

Pajares, Frank (2002). (Retreived 04,22,2005)
Overview of social cognitive theory and of self-efficacy.
http://www.emory.edu/EDUCATION/mfp/eff.html.

Pedagogisk forskningsinstitutt ved Universitetet i Oslo
Nettesidene til Pedagogisk forskningsinstitutt ved Universitetet i Oslo.
http://www.pfi.uio.no/uniped/gpm/pbl.html

Scheflen, A. E.; (1978)
Susan smiled: On explanations in family therapy.
Family Proceedings, 17, side 59- 68.

Schoenfeld, Alan H.; (1992)
Learning to Think Matematically: Problem Solving, Metacognotion, and Sense Making in Mathematics, i (Red.)Grouws, Douglas A.
Handbook of Research on Mathematics Teaching and Learning
Macmillan Education Ltd. New York, 1992 ss. 355-358

Sheard J. og Hagan D. L.; (1999)
A Special Learning Environment for Repeat Students,
Proceedings of ITiCSE '99, Cracow

Shipman Harry L., Duch Barbara J.; (2001)
The Power of Problem-Based Learning: A Practical "How To" for Teaching Undergraduate Courses in Any Discipline
Barbara J. Duch, Groh, Susan E. og Allen, Deborah E. (red)
ISBN**:** 1579220371
Stylus publishing

Sommerville, Ian; (2004)
Software Engineering, seventh edition
ISBN 0-321-21026-3
Pearson Education Limited, England

Stroustrup, Bjarne (1994)
The Design and Evolution of C++
Addison Wesley

Suchman, Lucy A.; (1994)
Plans and situated actions
ISBN 0-521-33739-9
Cambridge University Press

Tschudi, Finn; (1996)
Kvalitative metoder i samfunnsforskning
Harriet Holter og Ragnvald Kalleberg (red)
ISBN 82-00-22535-6
Universitetsforlagets Metodebiliotek

Williams, A. Laurie og Kessler, Robert R.; (2000 mai)
All I Really Know about Pair Programming I Learned in Kindergarten, 2000
Communication of the ACM

Woods, Don; (2005 okt.)
McMaster University
Hamilton, Ontario Canada
http://chemeng.mcmaster.ca/pbl/pbl.htm

Workshop on Learning and Teaching Object-orientation – Scandinavian Perspectives
Oslo, October 20. 2003.

Zeller, A.; (2000)
Making Students Read and Review Code.
SIGCSE Bulletin 32, 3, 89-92

VEDLEGG 1

Om Karel J

Introduksjon

Richard Borge

Når studenter starter på en utdannelse innenfor informatikk møter de raskt på et begrep som mest sannsynlig er ukjent for de fleste av dem: Objektorientert programmering.

Det var på slutten av 80-tallet at ”object-oriented” ble et slags moteord, men fortsatt var det stort sett avanserte kurs innen systemutvikling som drev med denne typen programmering. Det var fortsatt en del nøling med å introdusere objektorientert programmering i begynnerundervisningen [1].

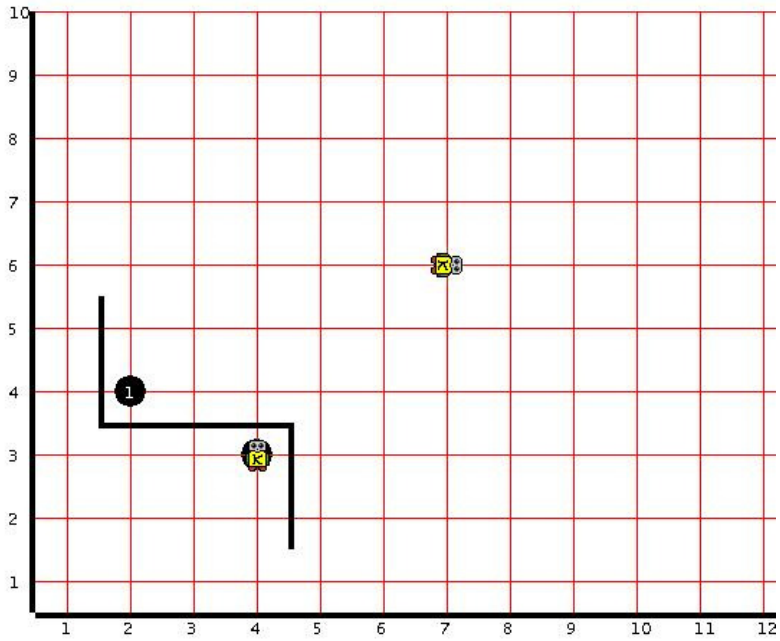
Det første objektorienterte språket het SIMULA og ble utviklet av Kristen Nygaard og Ole Johan Dahl på Institutt for Informatikk ved Universitetet i Oslo på midten av 1960-tallet. Språk som senere er brukt i undervisning er for eksempel Smalltalk, ADA og Blue [2]. Debatten på begynnelsen av 90-tallet gikk på om det er objektorientert (Java, C++) eller strukturert (Pascal, C) programmering som er det beste å lære først, et par eksempler er [3, 4]. I de senere år har stort sett det meste av programmeringsundervisning på høgskole og universitets nivå gått på Java og C++, men metodene for å lære bort en objektorientert tankegang har sittet fast i de gamle sporene, nemlig sporene til de strukturerte programmeringsspråkene. Objektorientert er ikke det samme som strukturert når man ser på programmering. Professorer verden over har forsøkt mange måter å lære bort objektorientert programmering på.

Visualisering (dvs. å bruke grafikk til å beskrive data eller et program) har lenge vært en metode for å lære bort programmering på. Seymour Papert introduserte tidlig på 1980-tallet språket LOGO for å lære barn å programmere [5]. LOGO var et strukturert språk, men tanken har blitt gjenfødt også i den objektorienterte verden, og jeg ønsker her å sette fokus på en av disse: Karel J.

Hva er Karel J?

Karel ble først utviklet av Richard Pattis og ble brukt til å lære bort Pascal til studenter i programmering (1981), men ble senere skrevet om til å lære bort C++ av Joseph Bergin, Mark Stehlik, Jim Roberts og Richard Pattis (1997) [6]. I 2000 og 2001 ble Karel++ (som var det C++ versjonen ble kalt) skrevet om til Java, og Karel J var født.

Karel er en robot, en liten todimensjonal robot som lever i en todimensjonal verden. I denne verdenen jobber Karel med å løse små og store oppgaver som den får gitt. Måten Karel løser disse oppgavene på er at den får et sett med instruksjoner skrevet i et programmeringsspråk fra sjefen sin. I disse tilfellene er språket Java og sjefen, det er deg. Som man ser av dette bildet har man mer enn kun roboter i denne verden. Man har vegger og små sorte rundinger som kalles for *pipere*. Det er med disse piperne og veggene Karel jobber med og manøvrerer rundt ved hjelp av forhånds gitte instruksjoner. Ved å lage komplette programmer i Java kan man fort løse eventuelle oppgaver, samt få en grafisk fremstilling av det hele på en måte som best kan beskrives som en hakket tegnefilm. Vi beskriver ingen direkte eksempler her, disse er beskrevet i materiell som deles ut til dere.



Hva kan Karel brukes til?

Da Papert introduserte LOGO, som dreier seg om å programmere en mekanisk skilpadde til å tegne, så han fort noe spennende: Når barna skulle programmere skilpadden apte de først bevegelsene skilpadden skulle gjøre for å få en forståelse for hvordan denne skulle gå. Papert så skilpadden som et objekt barna kunne tenke med og på denne måten få en forståelse for hvordan en instruksjon i en datamaskin kunne oversettes til bevegelse i skilpadden.

Tanken er beholdt i Karel: Roboten vår kan her bli tatt ut av en programmeringssammenheng og heller sees på som en robot man skal kommunisere med. Det eneste er at roboten snakker et språk som først må læres. Man kan på denne måten komme inn med programmeringstekniske detaljer uten å bruke et programmeringsteknisk språk, noe mange studenter har en tendens til å henge seg opp i: Mange tror det er viktig å huske begrepene, når det er teknikkene som begrepene beskriver som er det viktige. Samtidig får man en visuell fremstilling av hva en kommando i programmet ditt gjør og hvordan en endring kan få konsekvenser for oppførselen til roboten din.

Objektorientert tankegang kan være vanskelig å forstå i begynnelsen. Jeg må innrømme at objektorientert programmering var et mysterium for meg helt til jeg i avanserte kurs ble introdusert for objektorientert analyse og design. Karel J illustrerer tanken bak

objektorientert programmering meget bra: Man kan lett fremstille ulike roboter som objekter og på denne måten å skape mange ”knagger” som studentene kan henge begreper og tankeganger på uten å verken bli tekniske i språket eller abstrahere seg fra robotens enkle verden.

Hvordan kan Karel J best utnyttes?

Karel J kan gjøres ufattelig komplekst eller meget enkelt. Her viser Karel sin største styrke: Man kan introdusere Karel veldig tidlig; som i vår sammenheng, eller bruke det i avanserte kurs for å illustrere vanskelige eksempler. Forelesere kan velge å bruke Karel i et par forelesninger før de hopper over på mer generell programmering, eller de kan introdusere Karel når de trenger et verktøy for å illustrere et eksempel eller et begrep.

Vi føler at Karel best utnyttes dersom man først bruker Karel i et antall uker, basert på hvor store krav kurset som bruker programmet har, for så å flytte seg over til generell programmering. Men fortsatt bruker man Karel for å illustrere mer generelle begreper. På denne måten har studentene en verden de er kjent og fortrolig med, og et sett med ”knagger” der de kan henge begreper. Jeg bruker ordet ”knagger” en del fordi jeg føler at dette er et meget viktig aspekt av å lære: At man har et sted å plassere kunnskap, enten man bruker sanger for å huske ting eller todimensjonale roboter for å illustrere eksempler, er dette et av de beste verktøyene man har for å forstå og lære.

Referanser

[1]: Mark C. Temte: *Let’s Begin Introducing the Object-Oriented Paradigm,* ACM SIGCSE Bulletin 4, (1991)

[2]: Michael Kølling & John Rosenberg: *Blue – A Language for teaching Object-Oriented Programming,* ACM SIGCSE Bulletin 2, (1996)

[3]: Rick Decker & Stuart Hirshfield: *The Top 10 Reasons Why Object-Oriented Programming Can’t Be Taught in CS1*, ACM SIGCSE Bulletin 3, (1994)

[4]: Paul A. Luker: *There’s More to OOP than Syntax!,* ACM SIGCSE Bulletin 3, (1994)

[5]: Seymour Papert: *Mindstorms: Children, Computers and Powerful Ideas,* New York, Basic Books (1980)

[6]: Byron Weber Becker: *Teaching CS1 with Karel the Robot in Java,* ACM SIGCSE Bulletin 2, (2001)

Karel J

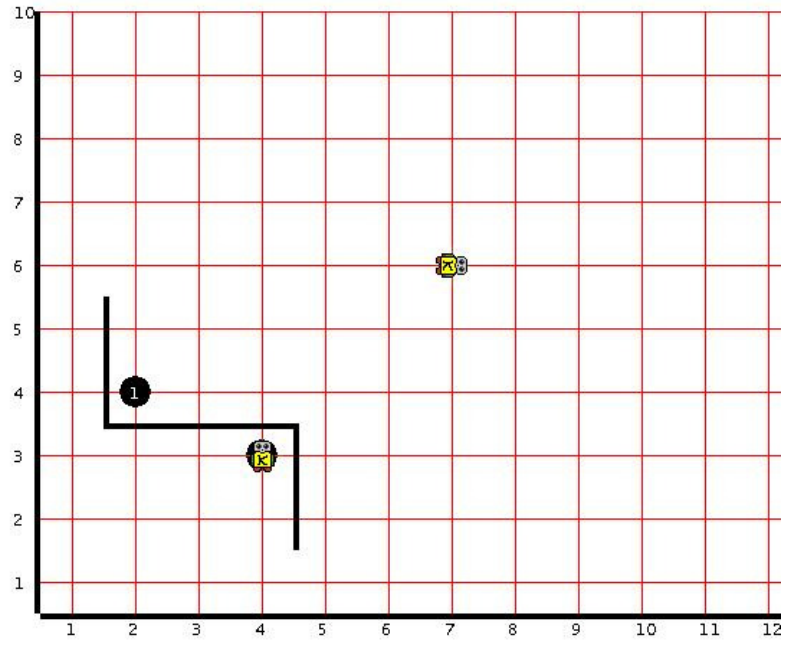
En aldri så liten innføring i robotkommunikasjon.

Roboten Karel

Karel er en liten robot som lever i en meget ukomplisert og enkel verden. Til tross for dette har Karel mer enn nok å gjøre i denne verdenen. Med stor entusiasme og innsatsvilje løser Karel alle de oppgaver som den blir gitt. Men som de fleste andre roboter er Karel rimelig dum og gjør kun det den er blitt bedt om på forhånd. Det er deres jobb å fortelle Karel hva den skal gjøre. For å i det hele tatt å kunne snakke med Karel er dere nødt til å lære dere å snakke språket hans. Det er et litt sært språk som ikke likner så veldig på noe dere har snakket før, men det burde gå fort å sette seg inn i den enkleste grammatikken. Men før vi begynner på språkskolen vår skal jeg fortelle litt om Karel og hans verden.

Karels verden

Karels verden er veldig symmetrisk og pent delt opp omtrent som New York, med Streets og Avenues.



Streets går fra vest til øst og Avenues går fra sør til nord. Resten av verden er sperret av slik at robotene ikke kan falle utenfor. Man ser at alle veier er nummerert veldig greit og enkelt fra 1 til 10. I denne verdenen jobber Karel med sitt. Dette arbeided dreier seg stort sett om pipere. Pipere er her de sorte små rundingene på kartet. Disse slipper ut en lien pipelyd (derav navnet) som Karel kan høre når han står oppå dem. Karel, som er den lille gule roboten med en K på brystet, går rundt og enten plukker opp eller legger ut pipere. På dette bildet har Karel fått hjelp fra sin gode venn Carol, som ser mistenkelig lik Karel, men det er bare en tilfeldighet. Noen ganger er Karels vei sperret av store sorte vegger som han ikke klarer å gå igjennom. Da må han navigere rundt, ellers så krasjer han i veggen og skrur seg selv av i panikk. Karel må følge alle gatene, han kan ikke gå på skrått, siden det er store bygninger i veien. Karel går ett kvartal av gangen, et kvartal er avstanden mellom to gater, og ender alltid opp i krysset mellom en Street og en Avenue.

Hvordan Karel navigerer

Karel navigerer med et kamera montert på hodet som kun kan se rett fremover og kan kun se et kvartal fremover av gangen. Det vil si at Karel kan oppdage en vegg før han krasjer i den. Samtidig kan Karel høre om han står på samme hjørne som en piper og

kan plukke opp disse med den mekaniske armen sin og legge dem i pipersekken sin. Han kan også ta pipere ut av sekken sin (dersom han har noen) og legge disse ut. Han kan bare plukke opp eller legge ut en piper av gangen. Karel beveger seg ved hjelp av instruksjoner gitt fra sjefen sin, og det er du som er sjefen hans. Siden Karel gjør kun hva han blir bedt om må du være veldig nøyaktig med hva du forteller Karel.

Karels språk.

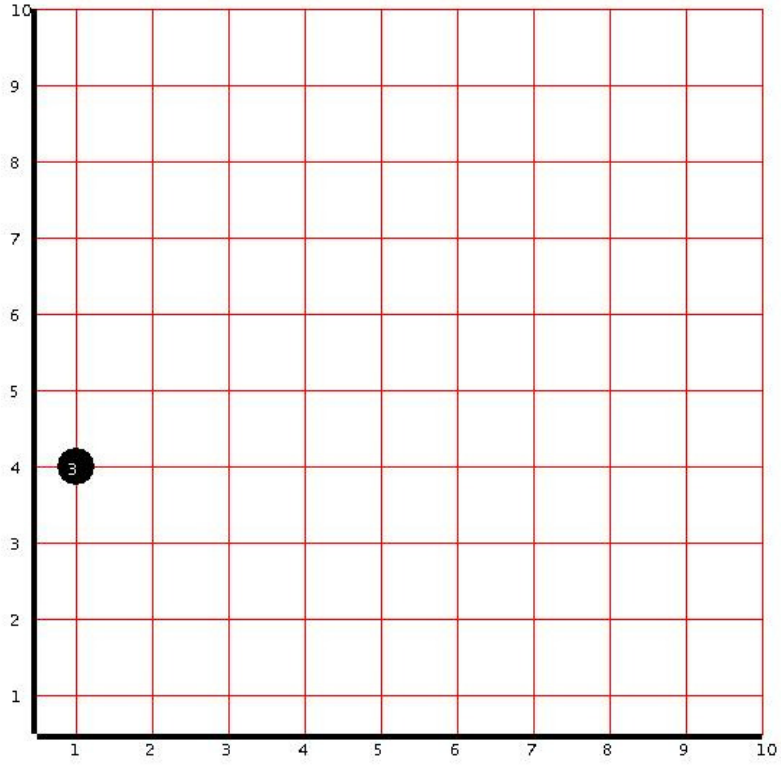
Karel har et meget begrenset vokabular og skjønner derfor bare en del ord. De enkleste av disse ordene er:

- **move();** Denne kommandoen forteller Karel å gå et kvartal frem.
- **turnLeft();** Denne kommandoen forteller Karel om å snu seg 90 grader til venstre.
- **pickBeeper();** Denne kommandoen forteller Karel å plukke opp en piper.
- **putBeeper();** Denne kommandoen forteller Karel å legge fra seg en piper.

Karel robotene produseres på Karel Werke, og må bestilles av deg. Disse robotene blir da laget og flydd ut med helikopter og plassert der du vil at de skal stå. Du må bare huske på å gi alle robotene dine forskjellige navn, samt hvor på kartet de skal stå, hvilken vei de skal se og om de skal ha noen pipere i sekken sin på forhånd. For eksempel: Dersom jeg vil ha en robot som heter karel plassert ut i krysset av 1. street og 2. avenue og som skal se nordover og ha null pipere i sekken sin, må jeg skrive:

ur_Robot karel = new ur_Robot(1, 2, North, 0);

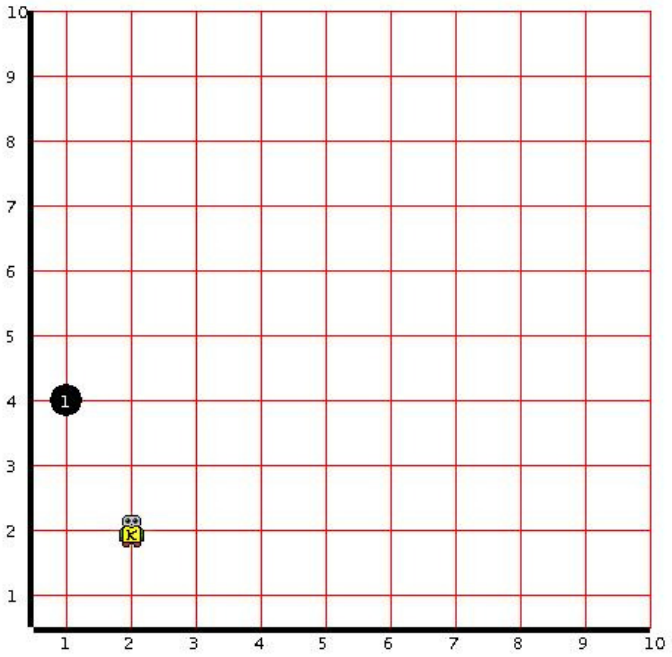
Nå har kanskje ting blitt litt forvirrende, så la oss se på et komplett eksempel. Alt som foregår er skrevet i språket Java. Dette er kartet vårt:



I denne oppgave skal karel starte i krysset mellom 2. street og 2. avenue. Han skal gå opp og plukke opp piperne som ligger i krysset mellom 1. street og 4. avenue. Først må vi skaffe oss en robot. Dette gjør vi ved følgende kommando:

ur_Robot karel = new ur_Robot(2, 2, North, 0);

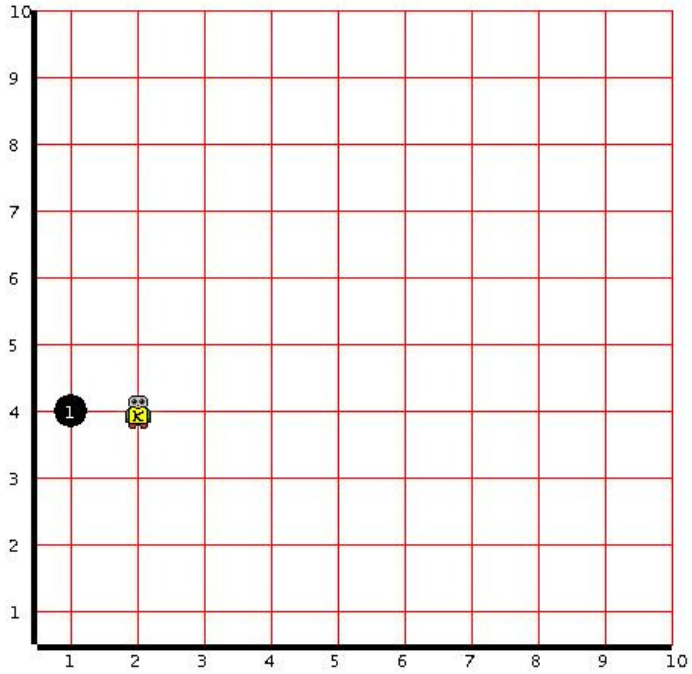
Nå ser situasjonen ut som følger:



Nå er det på tide å begynne å bevege på roboten, slik at vi kommer oss frem til piperen som skal plukkes opp. La oss gå to skritt frem. Dette gjøres ved å skrive:

karel.move();
karel.move();

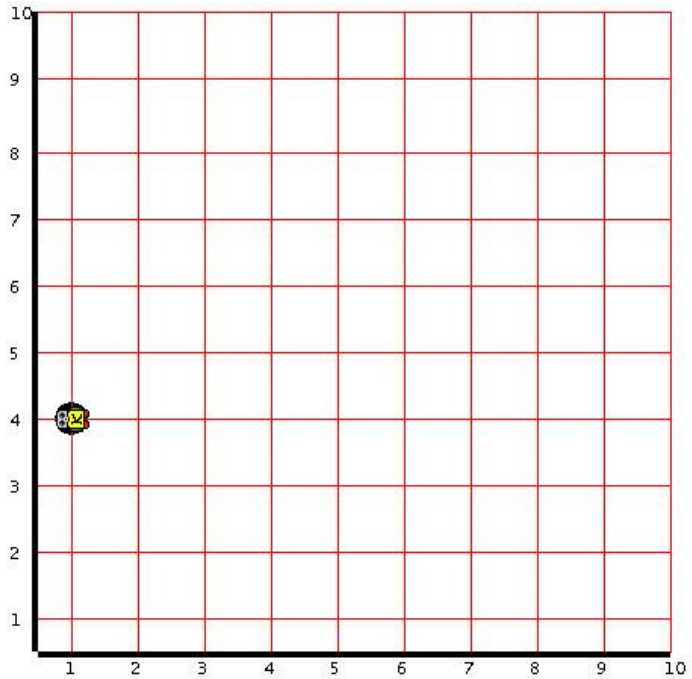
Vi husker fra tidligere at karel hadde et lite vokabular, og at move var et av ordene han forstod. Siden vi tilfeldigvis har kalt roboten vår ”karel”, må vi skrive karel.move() for at karel skal forstå et det er han vi snakker til. Når karel får beskjeden ”move”, vet karel at han skal gå ett kvartal frem i den retningen kameraet hans peker, i dette tilfellet nordover. Etter at vi har bedt karel gå to kvartaler nordover, ser situasjonen ut som følger:



Nå er det på tide å komme seg bort til piperen. For å gjøre dette må karel snu seg mot venstre (vestover) og gå et kvartal. Dette gjøres ved å skrive:

```
karel.turnLeft();
karel.move();
```

Da ser vi at situasjonen er blitt følgende:



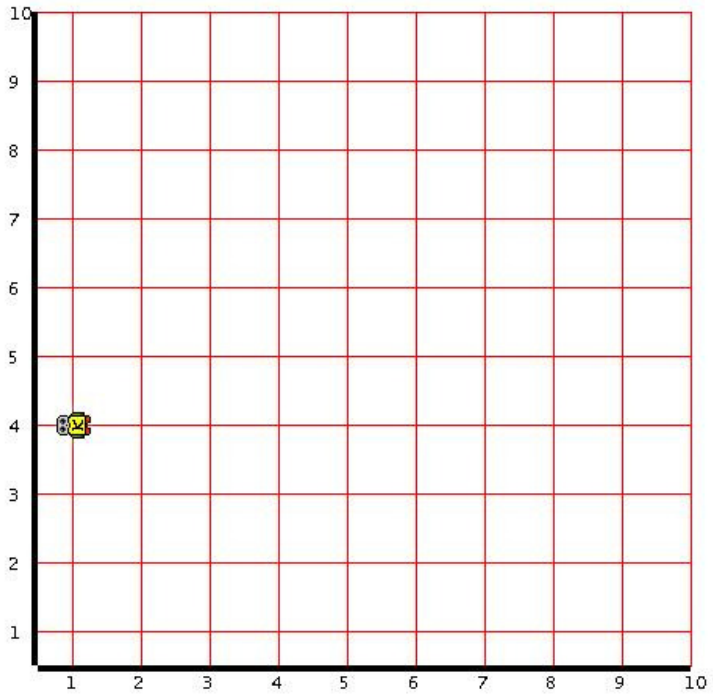
Nå står karel oppå piperen, og han kan plukke den opp og legge den i sekken sin. Dette gjør han da ved å skrive følgende:

karel.pickBeeper();

Fornøyd med vel utført arbeid kan karel hvile seg litt og spare strøm. Dette gjøres ved å skrive:

karel.turnOff();

Slutt situasjonen er da:



Vi har nå programmert vår første Karel robot, og jeg syntes vi gjorde en kjempejobb. Ut fra dette eksemplet er det ikke vanskelig å sette Karel til større og tyngre jobber.

Karel J

Neste skritt: Flere roboter

Karels gode venn

Vi skal nå introdusere Karels gode venn Carol. Carol er identisk med Karel, og det er jo ikke så rart, de er produsert på samme sted. Men med Carols hjelp går arbeidet til Karel dobbelt så fort fordi to roboter jobber bedre enn en (gammelt jungelord).

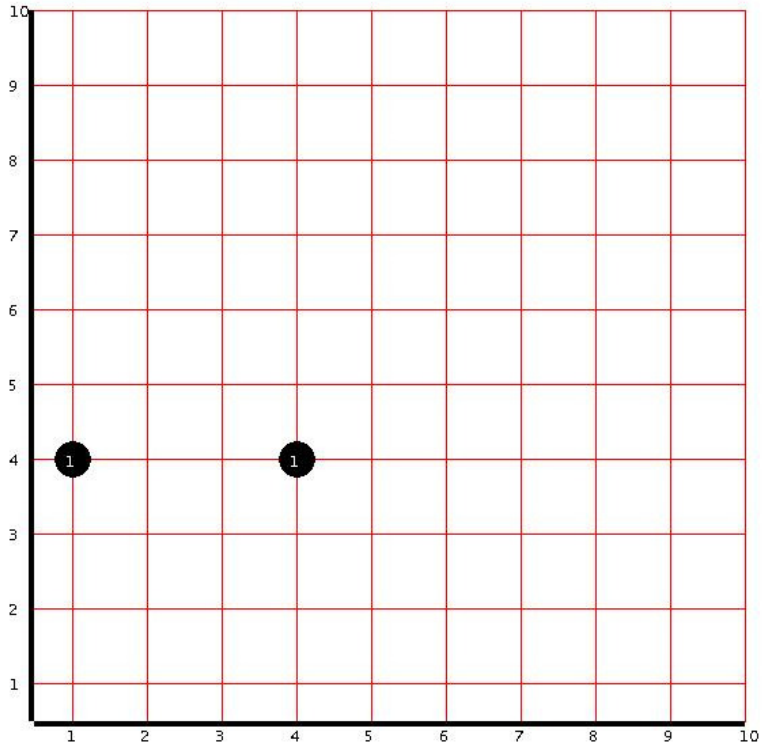
Å bestille Carol gjøres på samme måte som med Karel. Si at vi vil bestille en robot med navn ”carol” som står i krysset av 5.street og 4.avenue, ser mot øst og har 5 pipere i sekken sin. Da skriver vi:

ur_Robot carol = new ur_Robot(5, 4, East, 5);

Når vi nå skal be carol om å gjøre noe, må vi passe på at det er carol og ikke karel som får kommandoen, hvis ikke kan det bli mye krøll. Dersom du vil at både karel og carol skal bevege på seg, må du skrive:

karel.move();
carol.move();

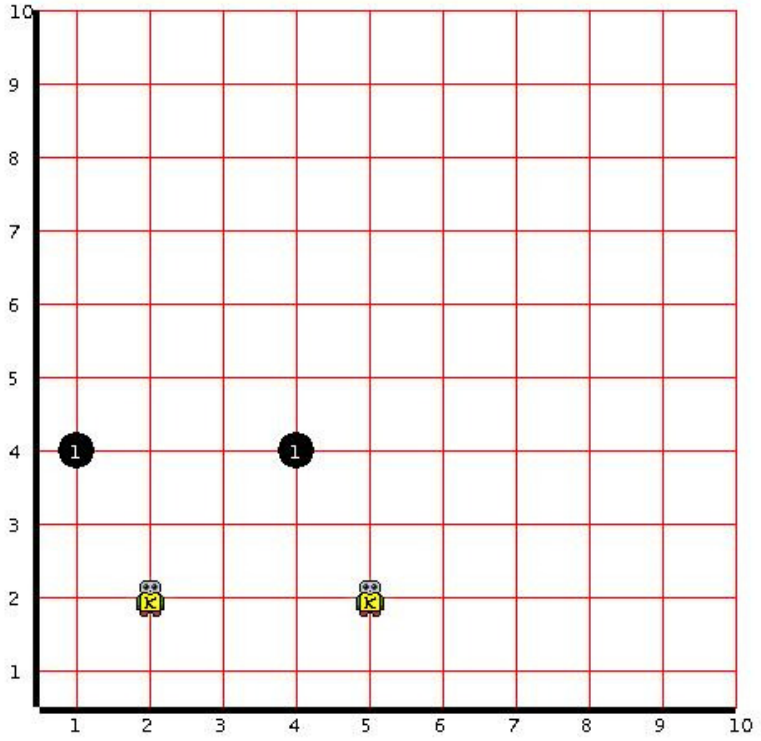
Her ser du at du først kaller på karel og gir ham en kommando å legge et punktum mellom karel og move, betyr at du kaller på karel sitt vokabular slik at det er kun karel som hører hva du har å si. Ellers ser programmet omtrent likt ut. La oss ta et eksempel, dette er kartet vårt:



Her har vi to pipere, og det er to roboter, karel og carol som skal plukke dem opp. Vi begynner programmet vårt med å bestille de to robotene. Klarer du å se ut fra kommandoene hvor de to robotene blir stående?

```
ur_Robot karel = new ur_Robot(2, 2, North, 0);
urRobot carol = new ur_Robot(2, 5, North, 0);
```

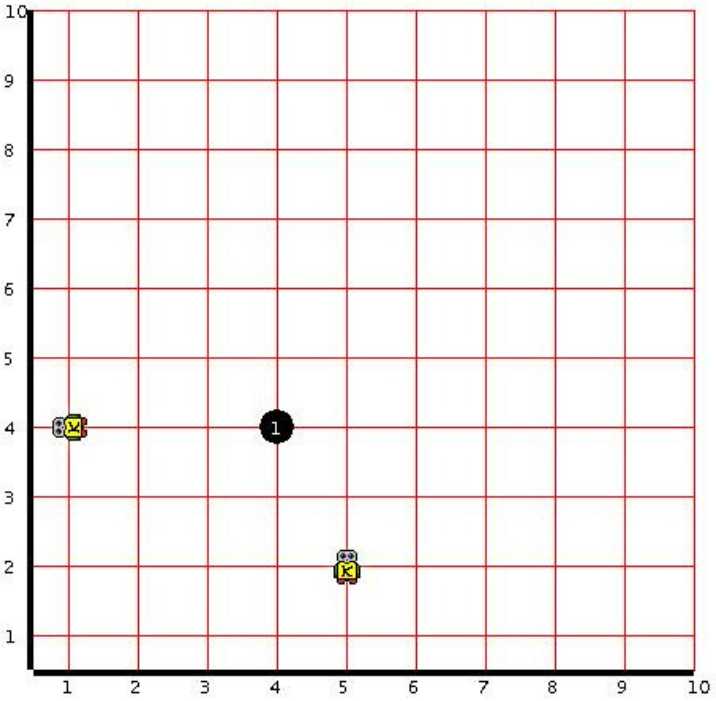
Da har vi følgende situasjon (karel til venstre, carol til høyre):



Man kan ikke be begge robotene jobbe samtidig. Kun en instruks kan bli sendt av gangen og denne kan kun bli sendt til en robot. Så la oss først la karel gjøre sin del av jobben. Da skriver vi:

```
karel.move();
karel.move();
karel.turnLeft();
karel.move();
karel.pickBeeper();
karel.turnOff();
```

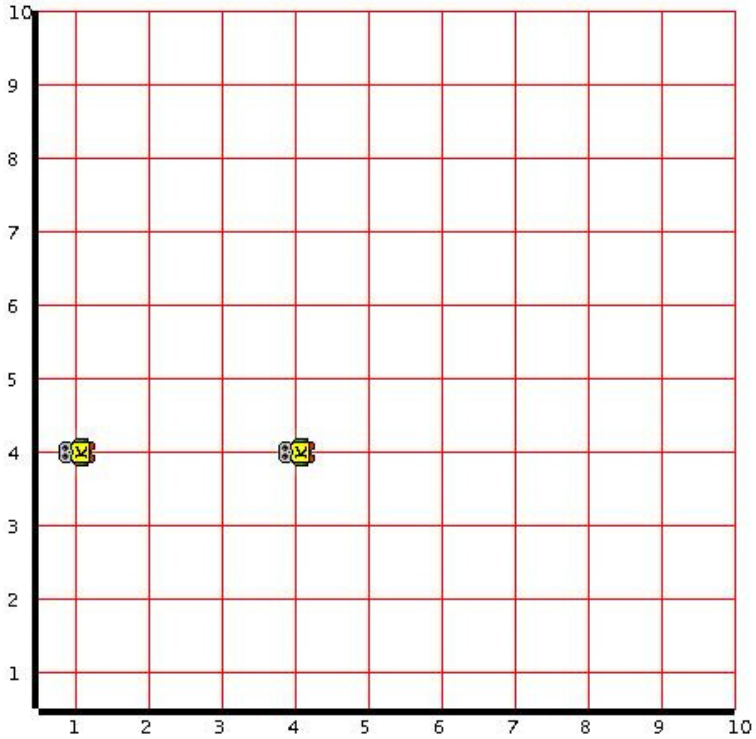
Dette fører da til at karel går hele løypen, lik den i forrige eksempel, mens carol bare står og ser på. Vi har da følgende situasjon:



Så er det carols tur. Her er det samme type instruksjoner som må til som for karel, men navnet må byttes:

```
carol.move();
carol.move();
carol.turnLeft();
carol.move();
carol.pickBeeper();
carol.turnOff();
```

Nå er situasjonen som følger:



Som man ser må man passe på å gi riktig navn slik at robotene skjønner hvem det er du prater med. Her kommer ”.” inn, som er en slags kobling mellom deg og den bestemte roboten. Vi kaller ofte dette punktumet for ”sin”. Se litt på det: Hvis vi sier kommandolinjen

carol.move();

med ord, blir det: ”carol sin move”, altså carol sin kommando ”move”.

Karel J 3

Subklasser og metoder:
Mer tilgjengelighet og oversikt.

Subklasser

I forrige eksempel så vi at det var to roboter som skulle gjøre noe. Karel og Carol skulle begge gå den samme ruten (noe forskjøvet). Det hele endte jo da med at man måtte skrive kommandoene for begge to to ganger, og dette er ikke en spesielt bra måte å skrive på. Ville det ikke vært kjekt å samle alle ting som roboter har til felles på et sted så vi ikke er nødt til å skrive slike lange serier for hver eneste robot? La oss introdusere en ny ting: Subklasser.

Vi har nå produsert flere roboter av typen ur_Robot:

ur_Robot karel = new ur_Robot(3, 5, North, 0);

Denne typen roboter har et vokabular som for eksempel **move()**. Vi har også muligheten til å lage våre egne typer roboter hvis vi vil med et eget vokabular. La oss prøve på det. Vi kaller denne modellen robot for PiperHenter:

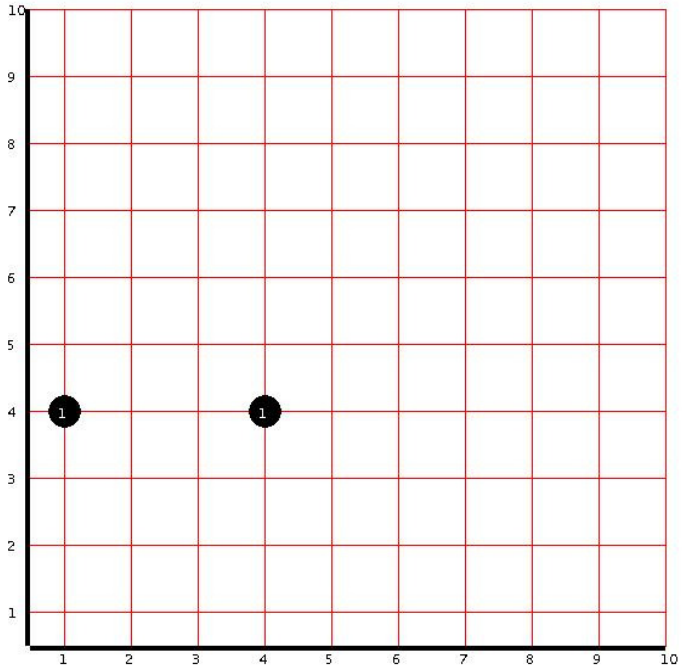
```
class PiperHenter extends ur_Robot {

    public void hentPiper() {

        move();
        move();
        turnLeft();
        move();
        pickBeeper();
        turnOff();

    }

} // Slutt på class HentPiper
```

La oss i første omgang se på det som står etter **public void hentPiper().** Vi ser her at vi bruker de samme ordene som i forrige eksempel med karel og carol. Det vi har gjort nå er å lage en ny type robot kalt PiperHenter og laget et ord som denne typen robot kan forstå, nemlig **hentPiper().** Dette ordet, **hentPiper()**, som karel kan forstå kalles en ***metode***. En metode er en forklaring på hva en robot skal gjøre når du gir den denne bestemte kommandoen, i dette tilfellet **hentPiper()**.

PiperHenter modellen som vi nettopp har laget er en underproduksjon av det mer kjente merket ur_Robot. Det vil si at PiperHenter har fortsatt alle egenskapene som ur_Robot har, men den har i tillegg noe ekstra som gjør den enklere eller bedre. I dette tilfellet er det **hentPiper()** som er tillegget.

La oss ta et eksempel:

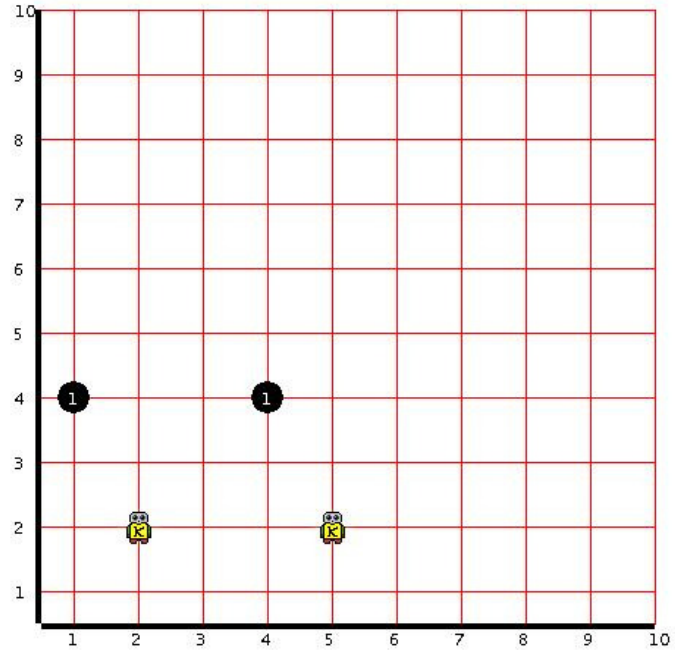
Vi har samme situasjon som fra forrige eksempel, det vil si:



Vi vil også plassere ut karel og carol på samme sted som i forrige eksempel. Det som er forskjellig er at nå skal vi bytte ut ur_Robot modellen vår med en ny modell robot. Denne roboten heter, som nevnt over, PiperHenter. Denne roboten har vi allerede sent in tegninga på til Karel Werke slik at de kan lage den for oss på bestilling. Disse bestillingene ser da ut som følger:

```
PiperHenter karel = new PiperHenter(2, 2, North, 0);
PiperHenter carol = new PiperHenter(2, 2, North, 0);
```

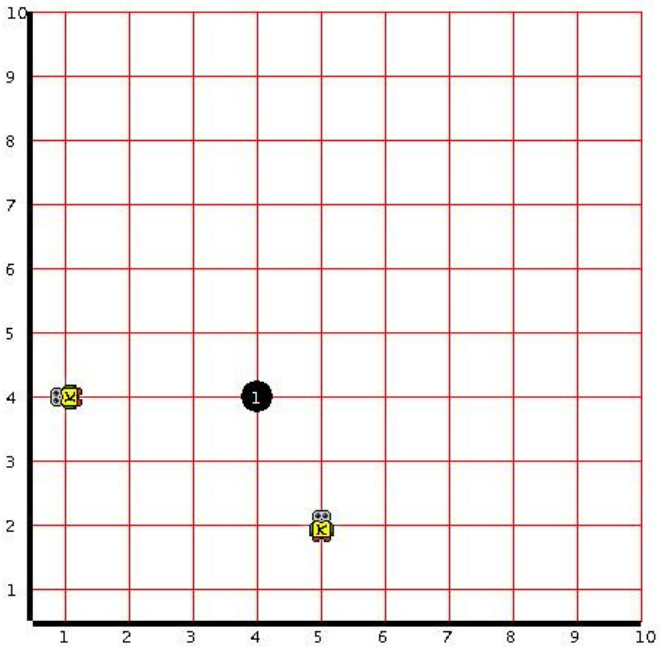
Da får vi følgende situasjon:



Vi husker nå at karel og carol ikke lenger er av modellen ur_Robot, men den nye og forbedrede PiperHenter. Vi har derfor et nytt ord i vokabularet til karel og carol, nemlig ordet **hentPiper()**. Så dersom vi nå skriver:

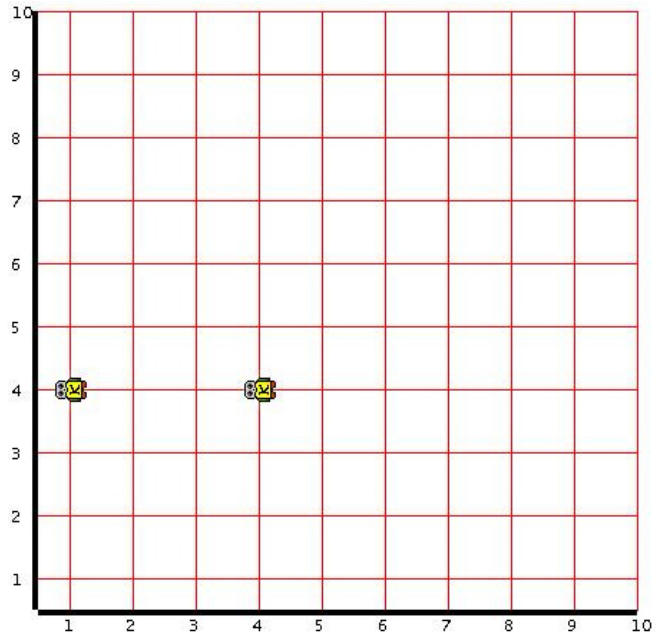
karel.hentPiper();

Så vil dette skje:



Så gjør vi det samme med carol:

carol.hentPiper();



Vi ser nå at med to korte kommandoer har vi fått begge robotene til å gjøre det de skulle. Vi ser i tillegg at i beskrivelsen av PiperHenter roboten trenger vi bare å skrive navnet på de ulike metodene som allerede er kjent i ur_Robot modellen. Dette er fordi PiperHenter modellen har arvet alle metodene fra sin tidligere modell.

Denne måten å dele opp forskjellige typer roboter på er en veldig fin måte å holde oversikten og enklere forandre på ting senere.

Karel J 4

Mer teknisk.

Noe mer teknisk

Vi har nå sett på en del om hvordan man lager roboter og spesielle typer av roboter, og hvordan man får dem til å gjøre enkle operasjoner. Men jeg tror strengt tatt at disse robotene aldri vil bli noe bra hvis ikke vi kan gjøre de litt mer intelligente og programmene litt mer effektive og kortere. Dersom vi skal bli tatt alvorlige som robotprogrammerere er det på tide at vi viser oss som mer profesjonelle.

Sannhetstester

Sannhetstester er faktisk akkurat som det høres ut: Vi sjekker om noe er sant eller usant og oppfører oss etter hva resultatet av denne sannhetstesten blir. For eksempel, dersom du har problemer med å skru på datamaskinen din, så gjør du små sannhetstester inne i hodet ditt. Først spør du deg selv: Er kontakten i veggen? Dersom svaret på dette er ”Ja”, fortsetter du deg med å spørre deg selv om strømbryteren bak på PC-en er skrudd på. Dersom svaret på dette er ”Nei”, skrur du på denne bryteren og ser om datamaskinen starter da, og så videre.

Karel har også noen slike spørsmål som han stiller seg. Igjen er disse spørsmålene uttrykt i et eget vokabular:

frontIsClear();	”Er det en vegg foran meg?”
nextToABeeper();	”Står jeg på en piper?”
nextToARobot();	”Er det noen robot ved siden av meg?”
facingNorth();	”Ser jeg mot nord?”
facingEast();	”Ser jeg mot øst?”
facingWest();	”Ser jeg mot vest?”
facingSouth();	”Ser jeg mot sør?”
anyBeepersInBeeperBag();	”Har jeg noen pipere i sekken min?”

På denne måten kan man sikre seg at Karel ikke løper på en vegg eller går i feil retning, noe som er ganske viktig. Så hvordan tester man disse tingene?

På robot språket vårt kalles disse testene for ”if-tester”. Hvorfor? Jo, en ”if-test” ser ut som dette:

```
if (Svaret på spørsmålet mitt er ”Ja”) {

    Gjør alt inne i denne krøllparentesen.

}
```

If betyr selvsagt ”hvis” på norsk. For å vise med et ordentlig eksempel:

Tenk deg at du har laget en robot ved navn ”karel” (dette er kjent stoff nå). Da kan en if-test se ut som dette:

```
if (karel.frontIsClear()) {

        karel.move();

}
```

Prøv å uttrykke dette med vanlige ord.

Hva om svaret på spørsmålet mitt ikke er ”Ja” da, men jeg fortsatt vil at roboten min skal gjøre noe? Da innfører vi et ord til: ”Else”. Else betyr ”ellers” på norsk. Så man kan lage noe som heter ”if-else tester” eller ”hvis-ellers tester”:

```
if (Svaret på spørsmålet mitt er ”Ja”) {

        Gjør alt inne i denne krøllparentesen.

}
else {

        Gjør alt inne i denne krøllparentesen.

}
```

For å ta et eksempel med ”karel” igjen:

```
if (karel.frontIsClear()) {

        karel.move();

}
else {

        karel.turnLeft();

}
```

Prøv å uttrykke dette med vanlige ord.

Det finnes en siste test også, som er ganske grei hvis man har forstått de to første. Den ser ut som følger:

```
if (Svaret på dette spørsmålet er ”Ja”) {

        Gjør alt inne i denne krøllparentesen.

}
else if (Svaret på dette spørsmålet er ”Ja”) {
```

Gjør alt inne i denne krøllparentesen.

}
else if (Svaret på dette spørsmålet er ”Ja”) {

Gjør alt inne i denne krøllparentesen.

}
.
.
.
else {

Gjør alt inne i denne krøllparentesen.

}

Det som er viktig her er at dersom flere av spørsmålene i denne lange testen gir svaret ”Ja”, er det fortsatt kun den første av testene som gir ”Ja” som blir utført.

Prøv å finne et godt norsk navn på denne testen!

Med disse sannhetsverditestene kan man gjøre robotene litt fleksible. Man er ikke nødt til å vite hvordan hele kartet ser ut på forhånd, og man kan unngå en del situasjoner. Men det er fortsatt rom for forbedring.

Gjentagelser

Tenk om vi må be karel gå 30 kvartaler. Det er 30 linjer der vi skriver **karel.move()** én gang for hver linje! Sånt går jo ikke an, man er da bedagelig anlagt. Vi må derfor finne en grei måte å gjenta oss selv på et visst antall ganger uten å måtte skrive så mye. Man har to måter å gjøre dette på: **for-løkker** og **while-løkker**.

For-løkker

For (sett startverdi; testbetingelse; inkrement) {

Gjør alt inne i denne krøllparentesen;

}

While-løkker

While (testbetingelse) {

Gjør alt inne i denne krøllparentesen;

}

Noen enkle kommandoer dere trenger

Her er noen enkle kommandoer som er nødvendig for å programmere. Skal ikke gå inn i detalj på glosene.

Dere kommer til å jobbe i et kommandovindu, kalt et ”shell”. Det er det samme som en MS-DOS ledetekst i Windows. Man jobber i dette ”shellet” ved å gi tekstkommandoer. De kommandoene dere bruker for å manøvrere rundt er:

cd <mappe>: Flytter til ønsket mappe.
ls: viser innholdet i mappen du er i.
mv <fil> <mappe>: Flytter en fil til en ny mappe.
emacs <filnavn>: Programmet vi bruker til å skrive i.

Eksempel: La oss si at jeg vil flytte meg et hakk ned i området mitt, til en mappe som heter ”karel”. Da skriver jeg bare ”**cd karel**” og trykker på enter. Dersom jeg vil flytte meg tilbake til der jeg var (dvs. ett hakk opp). Da skriver jeg ”**cd ..**” . Dersom jeg vil flytte en fil ”KarelEks1.java” til en mappe som heter ”eksempler”, skriver jeg ”**mv KarelEks1.java eksempler**”.

Dere trenger også noen flere kommandoer for å behandle programmene deres. Her kommer disse:

Kompilere et program

Å kompilere et program vil si å oversette programmet dere har skrevet til et språk datamaskinen forstår. Dette slipper dere å gjøre selv, dere ber simpelthen datamaskinen oversette. For oss er dette en litt lang besvergelse som ser ut som følger:

javac –d . –classpath .:KarelJRobot.jar <filnavn>

Dette er en litt vanskelig besvergelse, men trøsten er at dere ikke trenger å skrive den hver gang. Oppover-pilen på tastaturet blar igjennom alle deres tidligere kommandoer, så når dere har skrevet denne kommandoen en gang, kan dere bruke pilen og bare forandre på filnavnet.

Kjøre et program

Når programmet er kompilert kan dere kjøre dette, dvs. la datamaskinen gjøre det dere har bedt den om. Dette gjøres med denne besvergelsen.

java –cp .:KarelJRobot.jar karelther obot.<filnavn – java endelse>

Dette står forklart i en tekstfil kalt ”KompOgKjør.txt”. Skriv ”**more KompOgKjør.txt**”.

Vi skal også gi klar beskjed. Ikke heng dere opp i disse kommandoene, de er bare et nødvendig ”onde”.

VEDLEGG 2

Mal for å starte programmeringen

Mal for å starte programmeringen

```
// Lager egne pakker for bedre oversikt
    package karelTherobot;


    /*
     * Her er innpakningen til programmet. Husk å bytte ut navnet
     * under fra "DinFil" til noe annet.
     */
    class DinFil implements Directions  {


       /*
        * Under her starter programmet ditt. Når vi starter
        * programmet vil datamaskinen lete etter "main" og
        * starte alt derfra, så våre kommandoer må komme inni
        * der.
        */
      public static void main( String [] args )  {

         // Her kommer din kode:



      }
      // Gjør kartet synlig, sett inn navn på kartet under
      static  {


        World.readWorld( "mitt_kart.kwld" );
        World.setVisible( true );
       }
    }
```

VEDLEGG 3

Aktiviteter

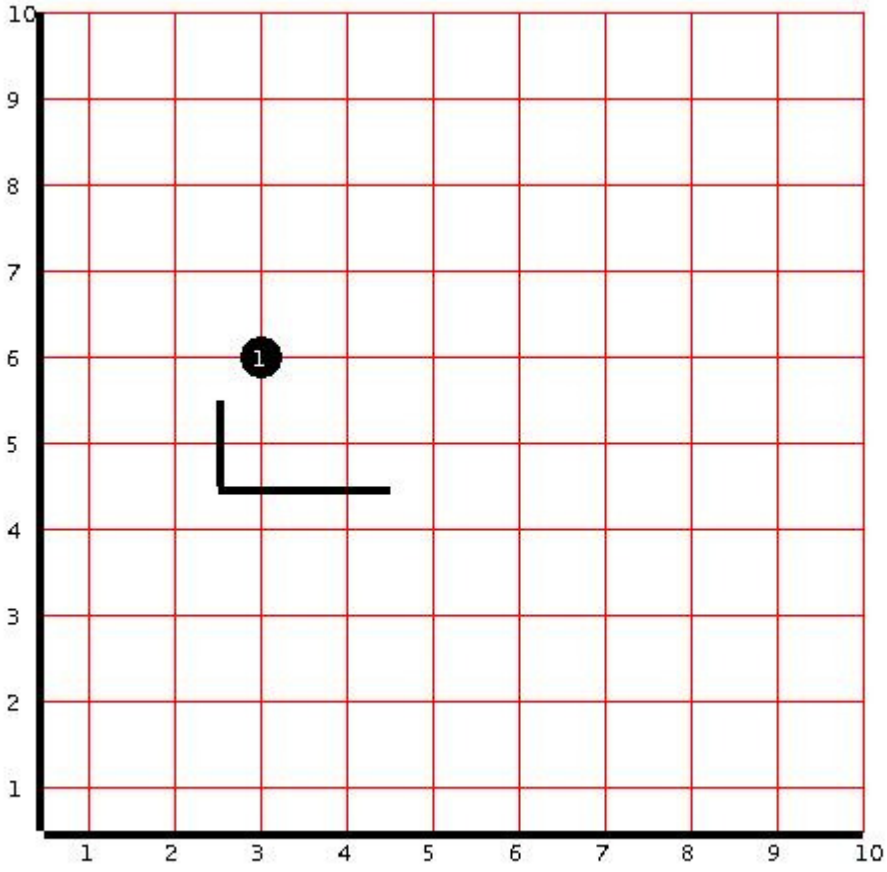
Aktivitet 1

Et objekt av typen ur_Robot kan bestilles fra fabrikken ved hjelp av:
ur_robot navn = new ur_Robot (gate, aveny, retning, pipere);

Objektet forstår bare noen bestemte kommandoer som du kan fortelle den.

I denne aktiviteten skal vi lære hvordan vi forteller objektet å komme seg forbi vegger og finne frem til piper. Objektet skal stå i krysset mellom 1. gate og 4. aveny.

Verden vi skal jobbe med her, ser ut som følger:

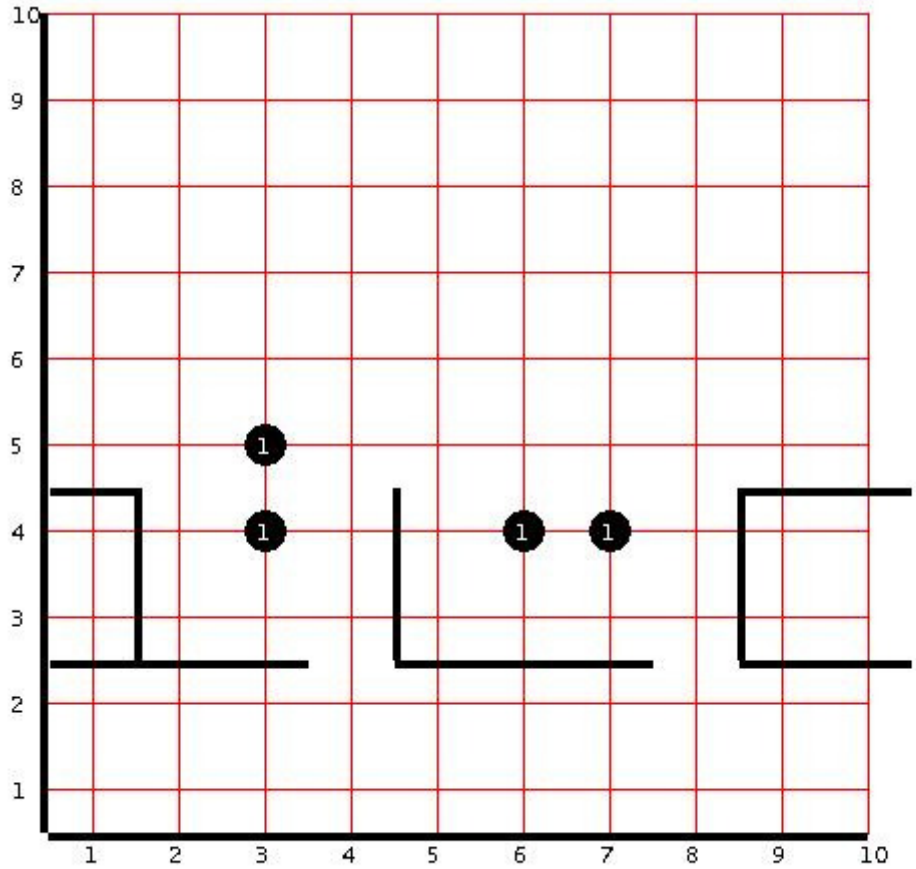


Aktivitet 2

Det er mulig å bestille flere objekter av typen ur_Robot, men du må skille disse ad ved å gi de forskjellige navn.

Det er navnet på objektet som brukes når man skal gi kommandoer til et spesielt objekt.

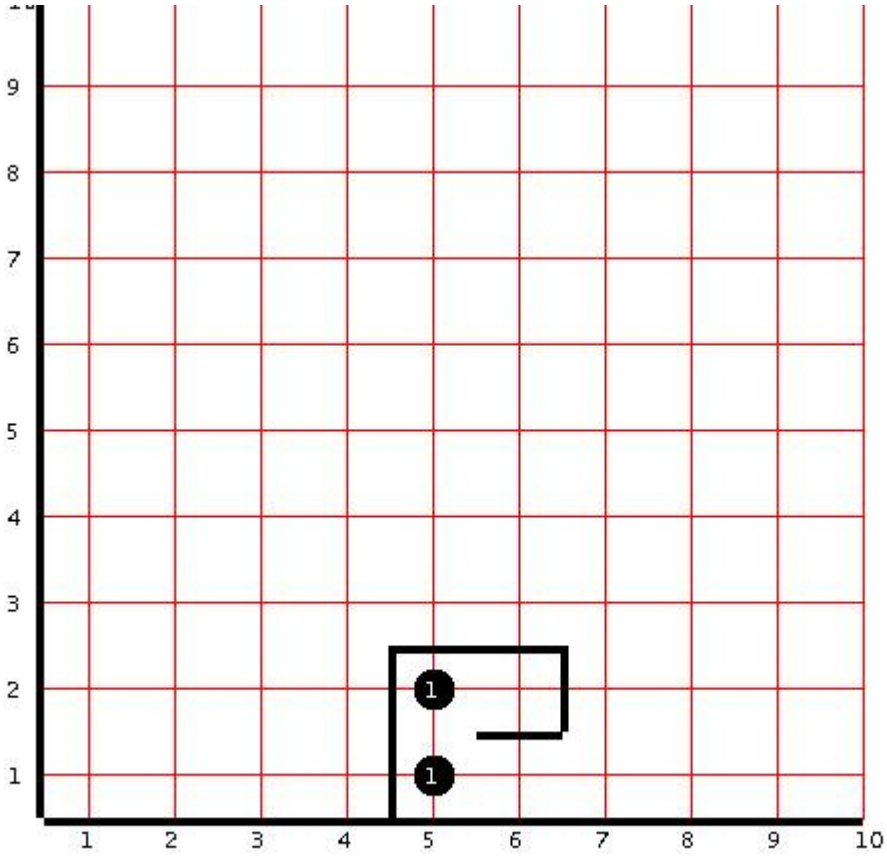
I denne aktiviteten skal vi la to objekter finne frem til hver sine pipere. Det første objektet skal stå i krysset mellom gate 1 og aveny 3, mens det andre objektet skal stå i krysset mellom gate 1 og aveny 7. Verden i denne oppgaven ut som følger:



Aktivitet 3-1

Vi har et problem: Objektene forstår ikke kommandoen **turnRight()**, som er kommandoen for å snu mot høyre. Som sett i tidligere oppgaver løste vi dette ved å snu til venstre tre ganger. Dette fører til mange ekstra linjer å skrive. Vi har mulighet til å bestille spesialiserte objekter fra fabrikken med egenskaper som vi har bestemt.

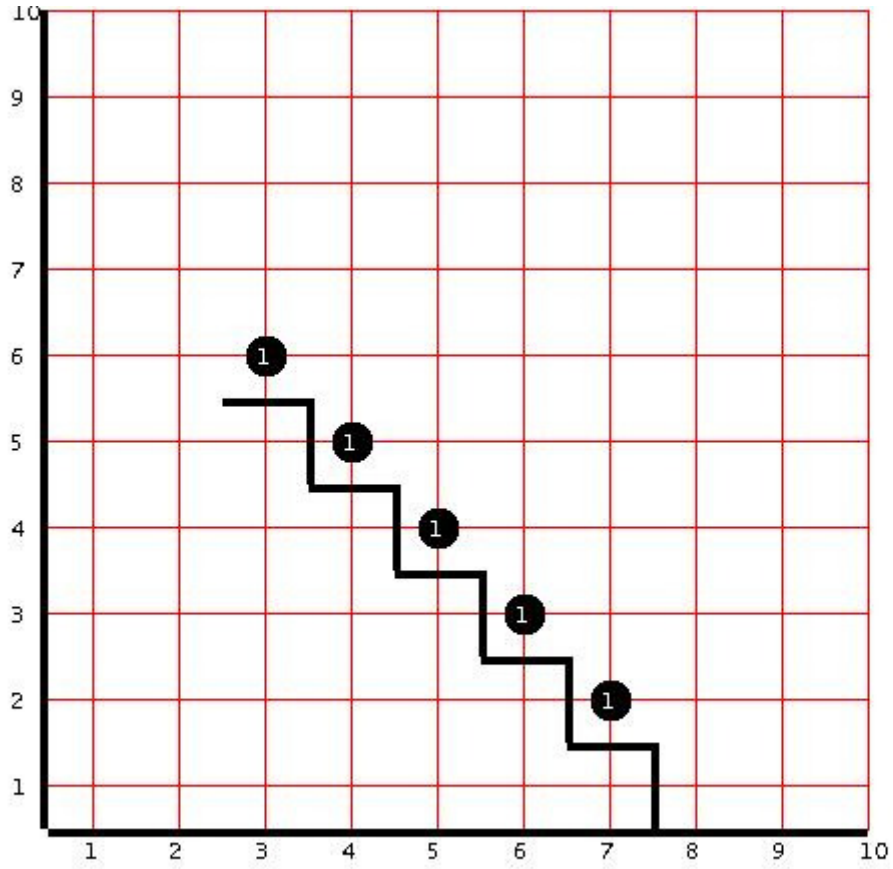
I denne oppgaven skal vi lage et spesialisert objekt av typen **ur_Robot2**. Dette objektet skal ha muligheten til å forstå **turnRight()**. Objektet skal starte i krysset mellom gate 1 og aveny 4. Verden ser ut som følger:



Aktivitet 3-2

I denne oppgaven skal vi la objektet plukke opp alle piperne i verden under. Problemet er at dette blir mange linjer å skrive. Hadde dette vært en trapp i en 10 etasjes blokk med en piper på hvert trinn og 15 trinn per etasje, ville dette tilsvart 750 linjer å skrive! Dette har vi ikke tid til, så vi må finne en bedre måte. Objektet skal starte i krysset mellom gate 1 og aveny 8.

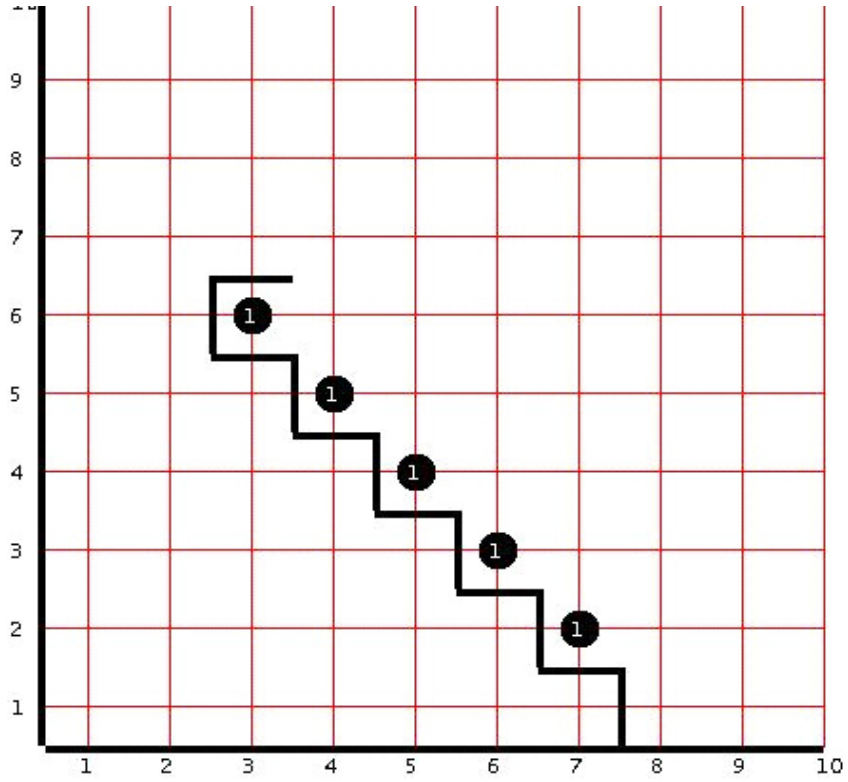
Hint: Prøv å finne et mønster i bevegelsene.



Aktivitet 4

Så langt har det vært du som har bestemt hvordan objektene skal forholde seg til verden rundt. Nå er det på tide å la objektene selv kunne ta hensyn til verden rundt seg. Objektene har fra fabrikken fått en del egenskaper som gjør at de kan finne ut litt om verden rundt seg.

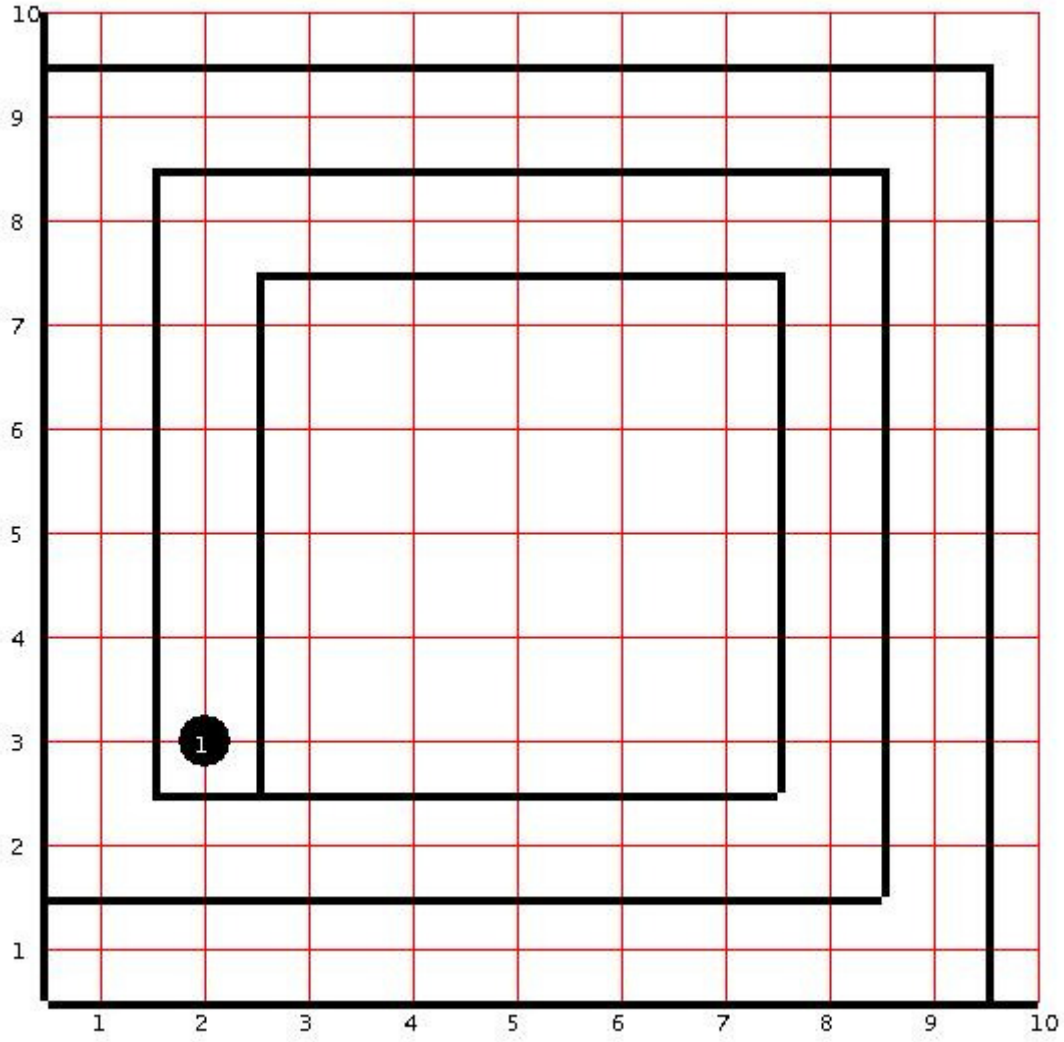
Denne oppgaven er en utvidelse av forrige oppgave. Objektet skal plukke opp alle piperne og stoppe før det krasjer i veggen mellom gate 6 og 7.



Aktivitet 4-2

I denne oppgaven skal objektet plukke opp en piper langt inne i labyrinten. Objektet skal starte i krysset mellom gate 1 og aveny 1. Verden ser ut som følger:

Hint: Løs dette på papir først ved å bruke vanlig språk.

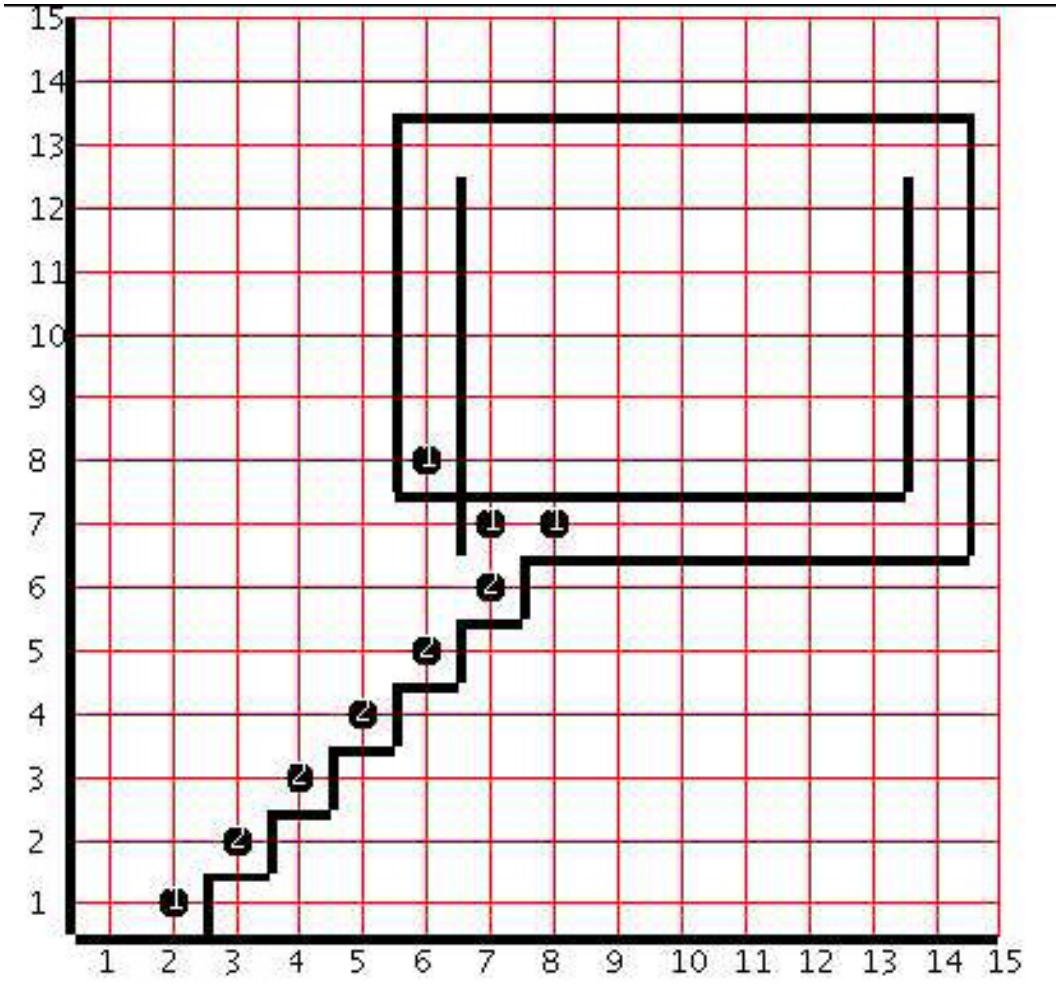


Aktivitet 5

Bruk samme kart som i aktivitet 4. Bruk den utleverte klassen, plukk opp pipere og skriv ut på skjermen hvor mange pipere Karel har plukket opp.

Aktivitet 6

Du skal lage 2 roboter. Den ene starter i (1, 2) og den andre i (8, 6). Begge robotene skal gå gjennom hele labyrinten og til slutt ha plukket opp 7 pipere. De får ikke lov til å plukke opp den piperen de starter på.



VEDLEGG 4

Oppsummering av aktiviteter

Oppsummering Aktivitet 1

I den første aktiviteten lærte vi hvordan man lager ett objekt av en type **ur_Robot**. Kommandoen for å lage et objekt er:

ur_Robot navn = new ur_Robot (gate, aveny, retning, pipere);

Vi lærte at dette objektet har et begrenset sett med kommandoer som objektet kan bruke. Disse kommandoene er definert fra fabrikken. De kommandoene vi lærte å bruke her er:

- **move();**
- **turnLeft();**
- **pickBeeper();**

Det vi også har lært er at objektet ikke forholder seg til ting rundt seg og gjør kun det som blir fortalt, selv om dette betyr å for eksempel krasje i en vegg.

Oppsummering Aktivitet 2

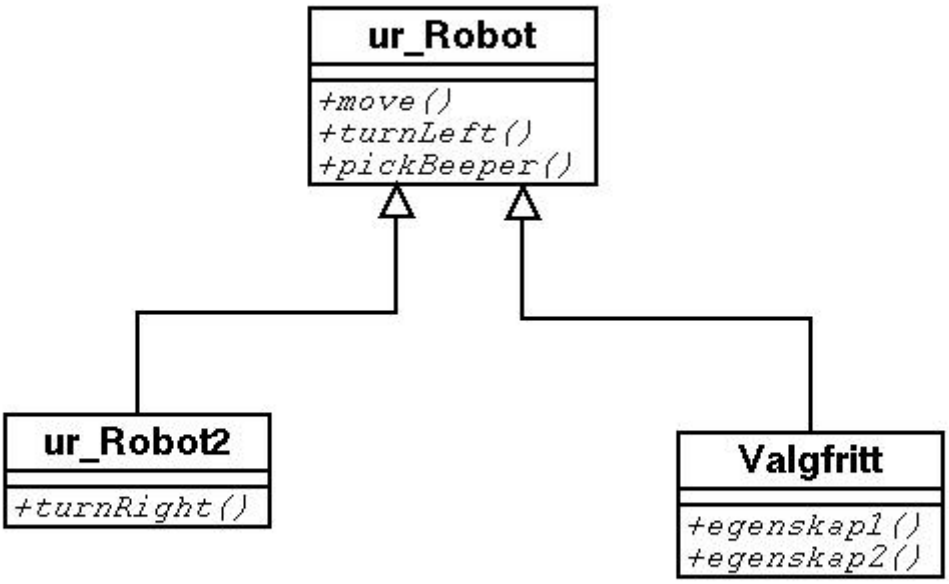
I denne aktiviteten lærte vi at vi kan lage flere objekter av samme type ur_Robot. Disse måtte skilles fra hverandre ved å gi de ulike navn. Vi så at selv enkle oppgaver medfører en del linjer.

ur_Robot er en *klasse*. En klasse er en generell beskrivelse av en mengde objekter i et system som har noe til felles. I vårt system, som er karels verden, har vi klassen ur_Robot som en generell beskrivelse av alle robotene. Objektene **karel** og **mini** har derfor de samme egenskapene fordi de begge kommer fra klassen ur_Robot.

Oppsummering Aktivitet 3

I denne aktiviteten lærte vi hvordan vi kunne lage spesialiserte objekter av typen ur_Robot som i tillegg til å arve ur_Robot sine egenskaper har sine egne definerte egenskaper.

Disse spesialiserte typene kalles for *subklasser*. For eksempel har subklassen ur_Robot2 alle egenskapene til ur_Robot pluss sin egen egenskap turnRight(). På tilsvarende måte har subklassen Valgfritt egenskapene til ur_Robot pluss sine egne egenskaper. Se figur neste side.



Oppsummering Aktivitet 4

I denne aktiviteten lærte vi at objektene har en del gitte egenskaper som gjør at de kan forholde seg til verden rundt seg. Det vi gjorde var å sette noen betingelser som måtte gjelde for at objektene oppføre seg på en spesiell måte. Vi introduserte repetisjon så lenge en betingelse var oppfylt.

Til å hjelpe oss med repetisjon har vi noe som kalles **while** og **if**:

- **while (betingelse er oppfylt) {**

  gjør alt innenfor krøll-parentesene;

  }

- **if (betingelse er oppfylt) {**
  gjør alt innenfor disse krøll-parentesene;
  }
  else {
  gjør alt innenfor disse krøll-parentesene;
  }

VEDLEGG 5

Om Robocode

Robocode Systemet

Dokumentasjonen som ble utlevert til studentene ved oppstart av aktiviteten.

Intallere:

Gjør følgende:

¿ mkdir robocode
¿ cd robocode
¿ cp richared/robocode-setup.jar .
¿ java -jar robocode-setup.jar
¿ ./robocode.sh

Dokumentasjon

Dokumentasjon finnes på
http : //robocode.alphaworks.ibm.com/docs/robocode/index.html,
men her er et sammendrag av noen enkle funksjoner:

Funksjoner som tilhører Robot

Disse funksjonene tilhører Robotklassen. Det vil si at dere kan kalle disse rett i programmet deres, siden deres robot er en subklasse av Robot og har derfor arvet all disse metodene.

ahead(double strekning)

Tanksen din flytter seg sa langt rett frem som er spesifisert i strekning.

Eksempelvis: $ahead(300)$ vil føre til at tanksen deres kjører fremover 300 (her er strekning gitt i 'steg' relativt til spillebrettet)

back(double strekning)

Det motsatte av ahead.

fire(double styrke)

Skyter en kule med en gitt styrke.

Eksempelvis: $fire(3)$ skyter en kule med styrke 3.

turnLeft(double grader)

Snur tanksen deres et gitt antall grader til venstre.

turnGunLeft(double grader)

Snur kanonen på tanksen deres et gitt antall grader til venstre.

onScannedRobot(ScannedRobotEvent ev)

Din tanks ser en annen tanks (i denne metoden blir da denne tanksen betegnet med en *e*). Du får i denne metoden mulighet til å programmere hva din tanks skal gjøre med dette. ScannedRobotEvent har en del metoder, som kommer under.

onBulletHit(BulletHitEvent bh)

Du treffer en annen tanks (med en kule som her har fått betegnelsen *bh*). Du får nå en mulighet til å programmere hva du skal gjøre når dette skjer. BulletHitEvent har en del metoder som kommer lenger ned.

ScannedRobotEvent

Hver gang din tanks ser en annen tanks, kalles denne metoden: $onScannedRobot(ScannedRobotEvent ev)$. I dette tilfellet fikk tanksen du så navnet ev. Nå kan man kalle en mange metoder ved å si:

$ev.metodenavn$. Metodenavnene kommer her:

getDistance()

Returnerer avstanden til den andre tanksen i forhold til din egen. La oss for eksempel si at du vil krasje med den andre tanksen (husk den andre tanksen har i dette eksempelet navn *ev*, se over). Da kan du skrive:

ahead(*ev.getDistance*() + 1). Dvs. kjør din egen tanks så langt som avstanden er + 1 slik at du krasjer.

getEnergy()

Returnerer livet til tanksen du har fått øye på.

BulletHitEvent

Hver gang man treffer en tanks, kalles denne metoden (dersom du velger å definere noe i den), f.eks.:
onBulletHit(*BulletHitEventbh*). I dette tilfellet fikk eventet (den andre tanksen) navn *bh*. Nå kan man kalle en mange metoder ved å si:

bh.metodenavn. Metodenavnene kommer her:

getName()

Returnerer navnet på tanksen du traff. Eksempelvis:

System.out.println("*Ihit*" + *bh.getName*() + "!!!"). Ikke bruk System.out.println (dette var bare et eksempel, bruk egne meldings metoder.)

Noen andre småting

Metodene over var bare noen eksempler, alt står på webadressen gitt (dette gir forresten glimrende trening I å lese en API!). Her følger noen ting dere burde kunne, men som jeg tar med så dere slipper å sitte å slå opp i boka deres i tilfelle :)

if-tester

If-tester brukes for å sjekke sannhetsverdier og utføre instruksjoner basert på disse sannhetsverdiene. Sagt på en mindre akademisk måte: Vi skal gjøre noe dersom det vi tester på er sant:

```
if(test = true) {
    gjør alt inne i klammene;
}
else if(denne testen == true) {
    gjør dette i stedet for;
}
else {
    ikke noe av det andre slo til, gjør dette;
}
```

En if-test trenger ikke å følges av en else. Det vil si at dersom ikke if-testen slår til, skal vi ikke gjøre noe spesiellt i det hele tatt. Eksempel på en Robocode if-test:

```
void onScannedRobot(ScannedRobotEvent ev) {

    // Vi er for langt unna, flytt nærmere
    if(ev.getDistance() > 500) {
ahead(getDistance()/2);
fire(1);
    }
    // Vi er for nærme, dra lenger bort
    else if(ev.getDistance() < 10) {
back(50);
fire(3);
    }
    // Avstanden er fin, bare skyt.
    else {
fire(2);
    }

}
```

for-løkker

For-løkker er fine når man skal gjøre noe et bestemt antall ganger. En for-løkke kan se ut som dette:

```
for(int start = 0; start < maxverdi; start++) {

    Gjør alt her inne;

}
```

Det vil si, så lenge start er mindre en maxverdi, gjør alt inne i krøllparantesene og øk start med en for hver gang.

while-løkker

While-løkker passer bra når man skal gjøre noe et uvisst antall ganger. Man har en eller annen testverdi, men denne er usikker på når slår til. En while ser typisk ut som:

```
while(så lenge dette er true) {

    gjør alt inne i krøllparantesene;

}
```

Vær forsiktig med while. Dersom man aldri får verdien false inne i while-testen, får man en evig løkke.

VEDLEGG 6

Programmeksempler

Listing fra tredje dagen
Fra gruppe 1:

Listing C.1: Code from group 1, Assignment 5

```
1  // Lager egne pakker for bedre oversikt
2  package karelTheRobot;
3
4  /*
5   * Her er innpakningen til programmet. Husk å bytte ut navnet
6   * under fra "DinFil" til noe annet.
7   */
8  class Karel52 implements Directions {
9      /*
10      * Under her starter programmet ditt. Når vi starter
11      * programmet vil datamaskinen lete etter "main" og
12      * starte alt derfra, så våre kommandoer må komme inni
13      * der.
14      */
15     public static void main( String [] args ) {
16         // Her kommer din kode:
17         TrappeRobot Karel = new TrappeRobot(8, 6, North, 0 , 0);
18         TrappeRobot Jon = new TrappeRobot(1, 2, North, 0, 0);
19
20         Jon.plukkTrinn();
21         Jon.turnRight();
22         Jon.finnPiper();
23         Karel.move();
24         Karel.finnPipe2();
25         Karel.turnLeft();
26         Karel.plukkTrinn2();
27     }
28
29     // Gjør kartet synlig, sett inn navn på kartet under
30     static {
31         World.readWorld( "eks6-1.kwld" );
```

```
32          World.setVisible( true );
33      }
34  }
35
36  class TrappeRobot extends Robot {
37      // Holder rede på hvor mange trinn vi har gått
38      int antTrinn;
39      // Konstruktør
40      public TrappeRobot(int gate, int aveny, Direction dir, int piper, int antallTrinn)
41          super(gate, aveny, dir, piper);
42          antTrinn = antallTrinn;
43      }
44
45      // Gå ett trinn
46      void ettTrinn() {
47          move();
48          turnLeft();
49          move();
50          pickBeeper();
51          turnRight();
52      }
53
54      // Snu til høyre
55      void turnRight() {
56          turnLeft();
57          turnLeft();
58          turnLeft();
59      }
60
61      // Økker telleren for antall trinn med 1
62      void leggTilTrinn() {
63          antTrinn++;
64      }
65
66      // Skriver ut antall trinn
67      void skrivUtTrinn() {
68          System.out.println( "Vi har gått " + antTrinn + " trinn.");
69      }
70
71      //Legger ut piper så lenge det er i sekken.
72      void leggPiper(){
73          while (anyBeepersInBeeperBag()){
74              putBeeper();
75              move();
76          }
77      }
78
79      void plukkTrinn(){
80          while (frontIsClear()){
81              move();
82              turnRight();
83              move();
84              turnLeft();
85              pickBeeper();
```

```
86          }
87      }
88
89      void finnPiper(){
90          while(!nextToABeeper()){
91              if (frontIsClear()){
92                  move();
93              }
94              else {
95                  turnLeft();
96              }
97          }
98          pickBeeper();
99      }
100
101     void finnPipe2(){
102         while(!nextToABeeper()){
103             if (frontIsClear()){
104                 move();
105             }
106             else {
107                 turnRight();
108             }
109         }
110         pickBeeper();
111     }
112
113     void plukkTrinn2(){
114         while (frontIsClear()){
115             move();
116             turnRight();
117             pickBeeper();
118             move();
119             turnLeft();
120         }
121     }
122 }
```

Listing fra tredje dagen
Fra gruppe 2:

Listing C.2: Code from group 2, Assignment 5

```
1  // Lager egne pakker for bedre oversikt
2  package kareltherobot;
3
4  /*
5   * Her er innpakningen til programmet. Husk å bytte ut navnet
6   * under fra "DinFil" til noe annet.
7   */
8  class Karel8 implements Directions {
9      /*
10      * Under her starter programmet ditt. Når vi starter
11      * programmet vil datamaskinen lete etter "main" og
```

```
12       * starte alt derfra, så våre kommandoer må komme inni
13       * der.
14       */
15      public static void main( String [] args ) {
16          // Her kommer din kode:
17          ur_Robot2 karela = new ur_Robot2(8, 6, North , 0);
18          ur_Robot2 karel = new ur_Robot2(1, 2, East, 0);
19          while (!karel.frontIsClear()) {
20              karel.trapp();
21          }
22
23          while (karel.frontIsClear()) {
24              if (karel.frontIsClear()) {
25                  while (karel.frontIsClear()) {
26                      karel.move();
27                      if (karel.nextToABeeper()) {
28                          karel.pickBeeper();
29                      }
30                  }
31              }
32              karel.turnLeft();
33          }
34          karel.turnOff();
35
36          while (karela.frontIsClear()) {
37              if (karela.frontIsClear()) {
38                  while (karela.frontIsClear()) {
39                      karela.move();
40                      if (karela.nextToABeeper()) {
41                          karela.pickBeeper();
42                      }
43                  }
44              }
45              karela.turnRight();
46          }
47          karela.turnLeft();
48          karela.turnLeft();
49          karela.move();
50          karela.pickBeeper();
51          karela.turnRight();
52          karela.move();
53          while(karela.frontIsClear()) {
54              karela.trapp2();
55          }
56      }
57
58      // Gjør kartet synlig, sett inn navn på kartet under
59      static {
60          World.readWorld( "eks6-1.kwld" );
61          World.setVisible( true );
62      }
63  }
64
65  class ur_Robot2 extends Robot {
```

```
66      public ur_Robot2(int gate, int aveny, Direction dir, int piper) {
67          super(gate, aveny, dir, piper);
68      }
69
70      void turnRight() {
71          turnLeft();
72          turnLeft();
73          turnLeft();
74      }
75
76      void trapp() {
77          turnLeft();
78          move();
79          turnRight();
80          move();
81          pickBeeper();
82
83
84      }
85
86      void trapp2() {
87          turnLeft();
88          move();
89          pickBeeper();
90          turnRight();
91          move();
92      }
93
94      void walkLong() {
95          while(frontIsClear())
96              trapp();
97      }
98  }
```