Hvordan fremme aktiv deltakelse i programmeringsforelesninger (digitale og fysiske) med opptil 500 studenter?

F. Skirbekk, *Norges teknisk-naturvitenskapelige universitet*

SAMMENDRAG: Gode programmeringsferdigheter er viktig innen svært mange fagfelt, spesielt innen ingeniørfag. Det er derfor avgjørende at de grunnleggende programmeringsemnene greier å gi studentene grunnlaget som kreves for å kunne benytte programmering som et nyttig verktøy senere i studieløpet og i jobb. De grunnleggende programmeringsemnene er gjerne store, med godt over 1000 oppmeldte studenter og forelesningsparalleller på rundt 500 studenter. Forelesningsutbyttet til studentene øker ved å fremme aktiv deltakelse i forelesningene, framfor at studentene blir sittende som passive tilskuere. Dessverre er aktiv studentdeltakelse ofte krevende å få til når det er flere hundre studenter til stede.

I støtteforelesningene i emnet TDT4102 Prosedyre- og objektorientert programmering ved NTNU har vi hatt stor suksess med å gi små oppgaver underveis i forelesningene som studentene har fått tenke på og diskutere med nabostudentene i 1–3 minutter før oppgavene besvares og diskuteres i plenum. Dette har ført til at samtlige studenter som er til stede i disse forelesningene deltar aktivt, noe som øker læringen for studentene og gjør det mer givende å forelese. Teknikken har blitt benyttet i både fysiske og digitale forelesninger. I digitale Zoom-forelesninger lønner det seg å bruke «polls» (avkrysning) for å hente inn studentenes svar dersom det er mange studenter til stede, slik at det ikke blir kaos i chatten.

1 INTRODUKSJON

Programmeringsferdigheter er viktige innen alle ingeniørfag (og veldig mange andre fagfelt). Grunnlaget for å bruke programmering som et verktøy senere i studieløpet og i jobb legges i de grunnleggende programmeringsemnene. Disse emnene er gjerne store, med godt over 1000 oppmeldte studenter og forelesningsparalleller på 500 studenter. For å sikre et godt utbytte av forelesningene bør studentene være aktive deltakere i forelesningen, framfor passive tilskuere.

Jeg har vært involvert som undervisningsassistent og vitenskapelig assistent i NTNU-emnet TDT4102 Prosedyre- og objektorientert programmering de siste fem årene. Dette er et stort emne, som de siste årene har hatt godt over 1000 oppmeldte studenter, og fungerer som en introduksjon til objektorientert programmering samt en videreutvikling av programmeringsferdighetene (de fleste) studentene har med seg fra IT grunnkurs. Som assistent i dette emnet har jeg vært med å holde totalt 55 støtteforelesninger, hvorav 19 har vært digitale over Zoom. Støtteforelesningene i TDT4102 fungerer som en ekstra gjennomgang av de delene av pensum som vi vet mange strever med, samt om tema som er ekstra relevante for fagets øvingsopplegg (øvinger = ukentlige programmeringsinnleveringer). Disse forelesningene består derfor både av teorigjennomgang og eksempler. I disse støtteforelesningene har vi hatt stor suksess med å gi små oppgaver underveis som studentene har fått tenke på og diskutere med hverandre før oppgavene besvares og diskuteres i plenum. Dette har ført til at samtlige studenter som er til stede i disse forelesningene deltar aktivt, noe som øker læringen for studentene og gjør det mer givende å forelese.

Hovedformålet her er å gi konkrete eksempler på hvordan aktiv læring har blitt fremmet i støtteforelesningene i programmeringsemnet TDT4102 Prosedyre- og objektorientert programmering, både for fysiske og digitale forelesninger, med opptil 500 studenter. Hva slags oppgaver har vært gitt underveis i forelesningen? Hvor lang tid har studentene fått til å tenke og diskutere? Hvorfor har det blitt gjort på denne måten? Vi har fått svært positive tilbakemeldinger på disse støtteforelesningene fra studentene (både via emnets avsluttende spørreundersøkelse i slutten av semesteret, og via referansegruppen), men det har ikke vært gjort noen studie på dette, så teknikken er kun erfaringsbasert.

2 AKTIV LÆRING I PROGRAMMERINGSFORELESNINGER

Metoden vi har benyttet oss av for å fremme aktiv deltakelse, og dermed aktiv læring, i støtteforelesningene består av tre steg:

1) Presenter en oppgave.
2) Gi betenkningstid/diskusjonstid.
3) Gå gjennom løsningen i plenum.

Disse tre stegene brukes gjentatte ganger i hver støtteforelesning, og bidrar dermed til å bryte opp teorigjennomgangen. Målet med disse oppgavene er at studentene aktivt skal bruke det de nettopp har blitt presentert av informasjon slik at de prosesserer informasjonen bedre og faktisk forstår det som har blitt gjennomgått. Min erfaring er at dette også øker mengden spørsmål studentene stiller, da det å prøve å løse oppgaver og tenke rundt temaene får studentene til å innse hva de ikke forstår og lure på hva som hadde skjedd dersom enkelte av oppgavens parametere hadde blitt endret. På denne måten blir det mer interaksjon mellom foreleser og student.

2.1 Oppgave – valg av oppgavetype og presentasjon av oppgaven

Å velge «rett» type oppgave er den mest krevende biten av denne teknikken. Det viktigste er at oppgavene ikke er for store, for da vil dette bli en for stor avsporing fra forelesningens egentlige fokus. Større programmeringsoppgaver bør heller tas i prosjekter eller innleveringer/oppgaver de løser utenom forelesningene. Kodeforståelsesoppgaver pleier å fungere veldig fint, da de gir mulighet til et lite dypdykk inn i en kort kodesnutt. Spørsmål om hvilken teknikk som bør benyttes i ulike scenarier og om hva man bør være obs på dersom man velger å benytte seg av en bestemt teknikk kan også være veldig vellykket. Dersom man har tenkt til å vise et live-kode-eksempel (altså at man koder «live» foran studentene for å vise hvordan man går fram) kan det i enkelte tilfeller fungere å formulere eksempelet som en oppgave først. Da kan studentene tenke/diskutere litt rundt hvordan de ville gått fram for å kode løsningen, før løsningen livekodes basert på studentenes innspill til hva man bør gjøre. Sistnevnte har jeg bare prøvd i mindre fysiske og digitale forelesninger (under 100 studenter), så jeg vet ikke om det vil fungere like godt i større forelesninger.

Oppgaven/spørsmålet bør alltid stå på lysarket slik at alle enkelt kan få det med seg, også de som ble distrahert akkurat i det foreleser presenterte oppgaven. I digitale forelesninger med mange hundre studenter lønner det seg å ha oppgaver der studentene kan krysse av for hvilket svar de tror stemmer i en poll (avkrysningsskjema, Zoom har polls man kan fylle ut og bruke) slik at det ikke blir kaos i chatten.

2.1.1 Eksempler

Eksempler på oppgaver som har vært gitt i støtteforelesningene i TDT4102 våren 2022 er presentert i *Fig. 1-5*. I TDT4102 er det programmeringsspråket C++ som benyttes.

Oppgave 1g, vår 2017: Hva skrives ut?

```
stack<int> intStack;
for (unsigned int i = 0; i < 10; i++){
    intStack.push(i);
}
intStack.pop();
intStack.pop();
cout << intStack.top() << ", ";
intStack.push(100);
cout << intStack.size() << ", ";
intStack.pop();
intStack.pop();
cout << intStack.top();
```

21

Fig. 1. En kodeforståelsesoppgave som er hentet fra ordinær eksamen 2017. Kodeforståelsesoppgaver fra tidligere eksamener kan ofte være gode utgangspunkt for oppgaver i forelesningene.

Templatefunksjoner

Hvilke krav må x og y oppfylle?

```
1  template <typename T>
2  //T er plugg-inn-typen
3  void mySwap(T& a, T& b){
4      T temp = a;
5      a = b;
6      b = temp;
7  }
8
9  void myFunc(){
10     // x og y defineres
11     mySwap(x, y);
12 }
```

7

Fig. 2. Kodeforståelse for å forstå begrensningene til templatefunksjoner.

Hvilken datatype skal vi bruke her?

```
1  ??? theAnswer = 42;
2  ??? oppfantBreakpoint = "Betty Holberton";
3  ??? programmeringErMorsomt = true;
4  ??? ukjent = 'x';
5  ??? boktittel = "The Travelling Cat Chronicles";
6  ??? pi = 3.14;
7  ??? krokodillerErDinosaurer = false;
8  ??? kulesteStudiet = "elkraft";
9  ??? utropstegn = '!';
10 ??? tegnMedDobbelFnutt = "x";
```

22

Fig. 3. Denne oppgaven ble gitt i en støtteforelesning i semesterets første uke. Den er veldig enkel, men siden den gis helt i starten av semesteret er det fortsatt mange som er usikre på enkelte av kodelinjene. Her er det også gjort et poeng ut av at enkeltfnutter og dobbeltfnutter betyr ulike ting i C++, mens det i Python (som er det de fleste studentene kan når de starter med TDT4102) betyr det samme.

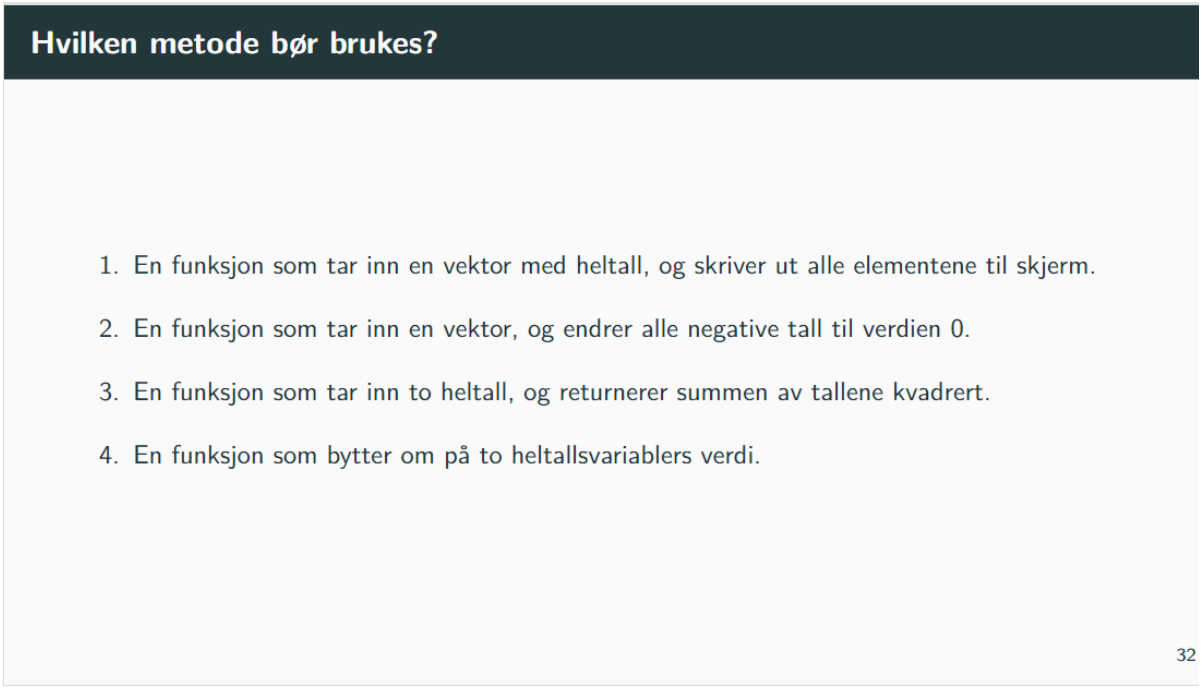

Fig. 4. En kortsvar/drøftingsoppgave om hvilken teknikk som bør benyttes. Denne forelesningen handlet om pass-by-value, pass-by-reference, og pass-by-const-reference, så spørsmålet er knyttet til hvilken av disse tre teknikkene som burde benyttes i de ulike tilfellene. Det interessante her er at det i enkelte tilfeller er flere man kan bruke, men likevel en teknikk som er foretrukket. Dette framprovoserer interessante diskusjoner og spørsmål.

Fig. 5. I denne oppgaven skulle studentene tenke på hvordan de ville gått fram for å løse problemet på lysarket. Løsningen ble deretter livekodet av foreleser basert på hva studentene foreslo å skrive. Dersom de foreslo noe som var feil var det lett å vise hvorfor det ikke fungerte. Her oppsto det interessante diskusjoner og mange gode spørsmål ble stilt. Dette var derfor veldig vellykket, men har bare blitt prøvd med færre enn 100 studenter til stede.

2.2 Betenkningstid – begrunnelse og forslag til lengde

Det er, slik jeg ser det, svært viktig å gi studentene litt tid til å tenke og til å diskutere oppgaven med de rundt seg. Dette gjør nemlig at alle får deltatt, uavhengig av om de tør å svare høyt i forelesningen eller ikke. På den måten sikres det at «alle» deltar aktivt, og dermed at alle har en aktiv læringsprosess. Som sagt er dette kun erfaringsbasert, og det er selvsagt ingen garanti for at samtlige studenter faktisk prøver seg på oppgaven, men basert på antallet oppmøtte studenter som får et ettertenksomt uttrykk i ansiktet mens de ser på lysarket, snakker sammen og peker opp mot oppgaven på lysarket virker det som majoriteten deltar aktivt.

Erfaringsmessig er det også flere som rekker opp hånden for å svare dersom de har fått litt tenke/diskusjonstid først. Sannsynligvis er dette fordi de både har rukket å tenke litt over problemstillingen før de blir tvunget til å svare, og fordi det er mindre skummelt å svare når du har diskutert med den ved siden av deg først (for da er du i hvert fall ikke alene om en potensielt feil løsning). I tillegg bidrar tenketiden til at det kommer flere spørsmål og oppfølgingsspørsmål når oppgaven gjennomgås da studentene har begynt å aktivt jobbe med teorien de har blitt presentert og dermed begynner å stille spørsmål ved hvorfor ting er som de er og hva som hadde skjedd dersom man hadde endret på en kodelinje.

1-3 minutters tenke/diskusjonstid er det som har fungert best i TDT4102. Dette er relativt kort tid, slik at studentene ikke rekker å spore av og begynne å snakke om andre ting. Samtidig er det lenge nok til at studentene har rukket å tenke litt rundt oppgaven og dermed både ha funnet forslag til løsning/del-løsning og ting de lurer på. Tidsrammen blir alltid informert om når oppgaven presenteres slik at studentene innser de at det er snakk om et kort tidsintervall og dermed ikke er tid til å ta pause.

I digitale forelesninger har jeg pleid å åpne pollen (avkrysningsskjemaet) når oppgaven presenteres, slik at studentene kan svare når de er klare. Pollen kan dermed holdes åpen i 1-3 minutter, slik at det kommer inn svar og studentene rekker å tenke litt før oppgaven gjennomgås. Selv om enkelte former for interaksjon blir borte i en digital forelesning, åpnes det også opp for interessante muligheter som for eksempel å få se hva majoriteten av studentene svarer på oppgavene. Dette kan gi verdifull innsikt inn i hvor mange som faktisk har forstått det som har blitt gjennomgått, og hva som eventuelt er de vanligste misforståelsene.

2.3 Gjennomgang av oppgaven

Etter at tenke/diskusjonstiden er over må oppgavene gjennomgås. Dersom det er en kodeforståelsesoppgave vil det ofte lønne seg å bryte det litt ned, og ta linje for linje. På den måten er det lettere for studentene å henge med på gjennomgangen, og det er mindre skummelt for studentene å svare siden de kun må forklare én linje. I en fysisk forelesning bør gjennomgangen i stor grad styres av studentene, og man bør få studentene til å rekke opp hånden og svare. Hvis studentene ikke er på ballen med en gang kan det ofte lønne seg å vente i 10-20 trykkende sekunder slik at studentene føler seg presset til å svare. Min erfaring er at denne ventingen blir unødvendig så snart en kultur for å svare på og stille spørsmål i forelesningene har blitt etablert.

I en stor digital forelesning får man ikke muntlige svar fra studentene (dette er både på grunn av GDPR og de tekniske problemene/kaoset dette fort hadde medført). Da blir man nødt til å gå gjennom oppgaven «alene». Men poll-svar (eller chat-svar) kan bidra til å gi et innblikk i hva som eventuelt bør forklares litt ekstra nøye – noe som er veldig nyttig.

Det er også viktig å åpne for spørsmål, både om temaet generelt og oppgaven, når oppgaven gjennomgås. Dette gjelder for både fysiske og digitale forelesninger. Basert på studentenes svar og spørsmål kan foreleser få verdifulle innblikk i hvilke deler av pensum som er vanskelig å forstå/bør forklares på andre måter enn det som har vært gjort til nå. Det kan også bidra til å avdekke kunnskapshull man ikke var klar over at studentene hadde.

3 AVSLUTTENDE ORD

Avslutningsvis vil jeg anbefale andre som foreleser grunnleggende programmeringsemner å prøve teknikken med oppgave-tenketid-gjennomgang for å øke studentdeltakelsen og studentenes aktive læring i forelesningene. Å øke studentenes læring burde være bonus nok i seg selv, i tillegg er det morsommere å forelese når man har et våkent og aktivt publikum. Det er heller ikke spesielt tidkrevende å implementere teknikken. Teknikken er trolig også overførbar til andre fagfelt.