Rapport Studentadministrasjon

I denne rapporten går vi kort gjennom framdriftsplanen for programmeringsprosjektet og ulike utfordringer vi opplevde. Vi fortsetter med å redegjøre for planleggingsprosessen, utforming av programmet og vurdering av pseudokode opp mot ferdig program. Til slutt har hvert gruppemedlem reflektert over eget bidrag i arbeidet, utfordringer de møtte på og eget utbytte av prosessen.

Framdriftsplan

Uke	Oppgave	Frister
44	**Øystein**: Forfiner valgt pseudokode, leverer. **Ida**: Utarbeider framdriftsplan **Alle**: Vurderer ønskede klasser for koding	Levering felles pseudokode Utarbeide framdriftsplan Levering framdriftsplan?
45	Endelig fordeling klasser for koding **Øystein**: Klasse "Filbehandling" **Anne Line**: Klassene "Main" og "Start" **Anders**: Klasse "Oppgave" **Vu**: Klasse "Student" **Ida**: Klasse "GenererLister"	Kommentar pseudokode fra lærere skal foreligge
46	**Alle**: Individuell koding / kort oppsummeringsmøte	
47	**Alle**: Individuell koding / samkjøring program	
48	**Alle: Kvalitetssikring av programmet ut fra vurderingspunktene i oppgaven.**	**28 November**: Levering individuelle bidrag til felles kode (minimum én klasse).
49	Eventuell justering/ferdigstilling	**5. Desember**: Levering kode Levering rapport (5-10 sider tekst, inkl ½ - 1 sider redegjørelse for hver enkelt students bidrag)

Kommentar på planen:

Planen vi utformet er i grunn fulgt til punkt og prikke. Samtlige gruppemedlemmer opplevde det som naturlig å jobbe en del individuelt med egne klasser etter innledende nettmøter og forfining av pseudokode. Ved spørsmål fungerte en Facebookgruppe som en noenlunde grei kommunikasjonskanal da det aldri var noe som hastet veldig, og derfor ikke krevde synkronkommunikasjon.

Kommunikasjon

Vi valgte å bruke Adobe Connect, appear.in og Facebooks gruppe- og chatfunksjoner som kommunikasjonsmidlene våre i arbeidet med denne oppgaven. Vi gjennomførte tre nettmøter hvor

vi evaluerte framgangen og diskuterte ulike problemstillinger vi hadde møtt på underveis. Dette fungerte særdeles godt, og ved behov for innspill underveis var det kort vei til de andre gruppemedlemmene gjennom Facebook. Framdriftsplanen la rammene for den forventede framdriften til programmet, hvilket ansvarliggjorde alle på gruppen; dette gjorde at planen ble overholdt på alle punkter, og til tider lå vi godt foran skjema også. Vi la møtene på slike tidspunkter hvor frister skulle overholdes, hvilket var med på denne ansvarliggjøringen, nettopp fordi det veldig raskt ville blitt tydelig hvis noen ikke hadde gjennomført sin del av jobben. Videre gav dette mulighet til å diskutere (og eventuelt forbedre) løsninger en har kommet fram til på egenhånd.

Utfordringer med å bestemme klasser, objekter og metoder og samordning av de enkelte programdelene

I planleggingsprosessen var programmets omfang vanskelig å se for seg (for novisene). Det var derfor, naturlig nok, en del diskusjon rundt hva som skulle skje i de ulike delene av programmet. Det var gjennom nettopp disse diskusjonene det ble klart for oss både hvem bruker skulle være og ikke minst hvordan programmet måtte bygges opp gjennom klasser og metoder.

Det ble tidlig klart for oss at det innad gruppen fantes store kompetanseforskjeller. Det var derfor litt usikkert hvor vi skulle legge ambisjonene for det ferdige produktet; for høyt kunne i verste fall resultere i en uoverkommelig læringskurve for nybegynnerne (hvilket videre ville betydd større arbeidsmengde for veteranen), mens for lavt ville bunnet ut i en relativt kjedelig oppgave for samtlige gruppemedlemmer. Vi bestemte oss derfor tidlig for å spille på hverandres styrker, og utformet derfor klassene til programmet spesielt med den hensikt at alle skulle få vise sitt kompetansenivå. Vi lyktes i noen grad i å oppnå dette; noen klasser er relativt små og enkle, mens andre bød på utfordringer som krevde stadige innspill fra gruppens samlede kompetanse.

Fordeling av klasser

Fordelingen av klassene gikk greit gjennom en spørreundersøkelse via Fronter som endte med at alle fikk det den klassen de ønsket seg. Gjennom arbeidsprosessen viste seg at det ble litt ujevn arbeidsfordeling da noen klasser var mer krevende enn andre. Dette er nok ikke til å unngå uansett, men vi løste dette greit med at de som programmerte mindre klasser hadde et større ansvar i rapportskrivingen.

Vurdering av pseudokode i forhold til ferdig programkode

I startfasen av arbeidet hadde flere gruppemedlemmer litt problemer med å tolke oppgaveteksten; var det student eller studentadministrator som skulle benytte seg av programmet? Som nevnt tidligere måtte det en del diskusjon til, og gjennom disse konkluderte vi med at bruker var studentadministrator. De første individuelle pseudokodene bærer preg av denne usikkerheten, og det var derfor naturlig for oss å gå for den mest komplette pseudokoden, nemlig Øysteins. Her kom det tydelig fram hva hovedfunksjonen i hver klasse skulle være. Dog ser vi i ettertid at vi ikke utnyttet potensialet som ligger i en forfinet pseudokode.

Klassen Filbehandling

Dette er i bunn og grunn en ganske enkel klasse. Den tar for seg to oppgaver; lagre et Gruppe-objekt til disken og hente et Gruppe-objekt fra disken. Operasjonen gir ikke muligheter for stor variasjon, og den endelige koden fulgte pseudokoden godt.

Klassen Gruppe

En særdeles enkel klasse som kun har i oppgave å oppbevare alle studentene, en pappeske å legge studentobjektene i om du vil. Dette slik at man enkelt kan sende alle studentene hit og dit i programmet og slik at lagringen kan foregå enklest mulig. Denne klassen har fulgt pseudokoden godt, sølv om det ble lagt til en ekstra metode som returnerer alle studentene som en array. Grunnen til dette var å gjøre det enklere å bruke bl.a. den innebygde Collections klassen til sortering.

Klassen GenererGruppe

Når det kommer til klassen GenererGruppe hadde denne alltid en klar hensikt; å generere ulike gruppesammensetninger. Oppgaveteksten er klar på at dette er en egenskap programmet skal ha, og det ble tidlig i planleggingsfasen klart for oss at vi ønsket å generere grupper med faktisk nytteverdi og som er praktisk anvendelige for studentadministrator.

Den første pseudokoden hadde fire ulike gruppesammensetninger; kjønn, alfabetisk, studiestart og tilfeldig. Denne pseudokoden var i aller høyeste grad råkode som beskrev hovedaktiviteten(e) i hver metode, og neste steg av forfining var derfor naturlig nok å beskrive aktivitetene i form av tenkte løkker og setninger. Den ferdige GenererGruppe-klassen stemmer overens med den første pseudokoden, men før selve kodingen startet gikk vi gjennom en relativt omfattende forfiningsprosess; nettopp for å klare skissere egenskapene til hver klasse på en god måte før vi startet med selve programmeringsprosessen. Dette var et bevisst valg ettersom vi ved å gå direkte på programmeringen hadde risikert å kaste bort tid på å kode områder i klassene som var uklare

gruppemedlemmene mellom; uklarheter som ble kartlagt og oppklart gjennom den skrittvise forfiningen av pseudokoden. Som et resultat ble vi som gruppe tryggere på hvordan selve utformingen av programmet skulle være. Da ble det også klarere for oss hvem brukeren skulle være – en studentadministrator. For å øke nytteverdien til programmet la vi derfor til muligheten til å sortere fagklasser. Videre ble det også klart at det var i denne klassen det var naturlig å hente ut studenter med alle arbeidskrav godkjent og videre kvalifisert til å gå opp til eksamen.

Klassene Start og Main

Klassen Start er den delen av programmet som setter det i gang, det vil si at det er denne som får programmet til å kjøre. Den har gått igjennom en liten forandring fra den tidlige pseudokoden hvor den skulle hente objektene fra Filbehandling og Gruppe, til å kun starte de ulike metodene i Main klassen. Dette var fordi det i begynnelsen ikke var helt klart hvordan denne typen initialisering fungerte. Til slutt i det kjørende programmet er Start ganske så lik som den var i den tidlige pseudokoden, med unntak av at det den kaller inn har litt andre navn.

Main-klassen har som oppdrag å hente inn alt av data som programmet så skal bruke senere, enten dette gjelder tidligere data som er lagret eller ny data som brukeren skal legge til. Denne klassen har endret seg drastisk fra den tidlige pseudokoden, der vi først hadde forestilt oss at den kun skulle hente inn tidligere studenter som brukeren hadde skrevet inn for så å gi brukeren en mulighet til enten å legge til nye studenter eller generere lister. I den siste kjørbare utgaven starter klassen likt som i pseudokoden; med å hente inn nye student-objekter, for deretter å starte en meny som gir brukeren tilgang til et sett med valg. Valgene brukeren får er å legge til nye studenter, generere grupper, endre studenter eller legge til oppgave, eller å avslutte hele programmet. Alt etter hva brukeren velger å gjøre så settes det i gang en metode, enten innad i klassen Main eller i andre klasser som blir kalt opp i Main. I pseudokoden vår hadde vi ikke bestemt oss for hvilke typer med input brukeren skulle komme med, men oppgaveteksten hadde tydelige krav i tillegg til at vi ville ha mulighet til å generere ulike grupper, noe som gjorde at vi måtte ha med en del input fra brukeren. Det vi så på som nødvendig i forhold til oppgaveteksten og det som programmet senere skal gjøre, var å få fornavn, etternavn, kjønn, studiestart og fagområde fra brukeren. Denne informasjonen ble senere sendt til klassen Student for å midlertidig lagre det der og deretter sende det inn i klassen Gruppe for å lage en array som inneholder dataene som objekter. I begynnelsen hadde vi forestilt oss at Main skulle kalle opp en return verdi fra klassen GenererGruppe og siden skrive ut til skjerm. Dette viste seg å være litt komplisert å få til, siden det var veldig mange return-verdier og at disse fort gikk litt i surr. Derfor starter kun Main en metode i klassen GenererGruppe som da også skriver ut en beskjed til brukeren om hvordan disse grupperingene så ut.

En av egenskapene til Main er at uansett hva brukeren velger så vil han/hun enten få beskjed om at det er skrevet inn noe feil, eller så vil programmet avslutte. For eksempel hvis brukeren lar en input-String stå tom så vil brukeren få beskjed om at vi må ha data der for å kunne fortsette. Da kan brukeren igjen velge å prøve på nytt eller avslutte programmet ved å trykke på x eller cancel. I tillegg til dette kjører Main en del tester på input fra brukeren for å dobbeltsjekke at vi faktisk får den type data som vi ønsker. Igjen, hvis brukeren skriver noe feil får den beskjed om dette. Som alle andre type programmer vil vi gjerne at programmet skal starte på nytt når man har fullført et av valgene slik at man kan foreta seg nye valg uten å måtte starte programmet på nytt. Dette ligger også inne som mulighet, i og med at programkoden er omsluttet med en do/while-løkke.

Klassen Oppgave

I oppgaveteksten ble kravene til løsningen skissert, og den skulle blant annet gi informasjon om hvilke oppgaver (arbeidskrav) studentene har levert, hvilke oppgaver de har fått tilbakemelding på, og om de kvalifisere for å gå opp til eksamen. Klassen oppgave er svært sentral i denne sammenhengen, da den tester om studentene har fått sine oppgaver og arbeidskrav godkjent, og følgelig «godkjenner» om de kan gå opp til eksamen.

Beskrivelse av klassen korte trekk: Dette er en forholdsvis enkel men viktig klasse. Klassen inneholder informasjon om en enkelt oppgave studenten har gjort. Dato for når oppgaven ble levert, om det er gitt tilbakemelding, om den er godkjent. Klassen mottar input fra andre klasser gjennom en konstruktør og utfører set og get metoder for når oppgaven ble levert, om det er gitt tilbakemelding og om den er godkjent.

Etter tilbakemelding på felles innlevering nr. 2 av pseudokode fikk vi tilbakemelding på at noen av klassene burde forfines, deriblant Oppgave. Det ble laget en ny pseudokode for denne klassen. Denne ble ytterligere forfinet til et tredje nivå når jeg begynte å programmere klassen. Klassen har vært grei å utvikle. Forfiningen etter revisjon tre av pseudokoden hjalp meg med utviklingen av klassen. Klassen kjørte uten store problemer da den ble testet sammen med de andre klassene i løsningen.

Klassen Student

Vu startet arbeidet med denne klassen, men trakk seg mot slutten av november. Øystein tok over kodingen av klassen.

Klassen fulgte pseudokoden med noen få unntak. Metoden “hent navn”, som skulle kombinere fornavn og etternavn for så å returnere denne strengen, ble erstattet med .toString(). Det fordi mange av Java sine innebygde klassen bruker toString() metoden for å hente ut informasjon, f.eks. ved å legge et studentobjekt til en liste, så vil Java bruke toString metoden for å avgjøre hvordan

objektet blir vist. ID metoden som i følge pseudokoden skulle generere en tilfeldig unik verdi til hver student (i tilfelle flere studenter med samme navn), ble erstattet av studentnummer. Vi utelot å implementere studentnummer ytterligere da det ikke var krav om dette i oppgaven. Ettersom variablen ligger klar, så vil en senere benyttelse ikke bryte kompatibilitet i forhold til serialiseringen.

Vi oppdaget også at hentKjønn() ville være upraktisk, og endret denne til isMann() som returnerte en boolean i stedet. I likestillingens navn la vi også til metoden isDame().

Resultat og eventuelt videre arbeid

Vi hadde et kjørbart program allerede to uker før innleveringsfrist. Dette opplevde vi som svært nyttig da det gjorde det lettere for oss å identifisering ulike glipper i programmets logikk, og lot oss gjennom dette øke brukervennligheten og programmets logiske oppbygning.

Vi ser også at det er stort potensiale for å utvide og forbedre programmets funksjon ytterligere. Det første elementet som ville vært naturlig å forbedre er å gjøre hele programmet vindusbasert. Bakgrunnen for at vi velger å levere en blandet løsning er nettopp for å ivareta identiteten til hvert gruppemedlem.

Det ville også være nyttig å legge til et fjerde valg i startmenyen, hvor bruker får mulighet til å skrive studentinformasjonen til fil, gjennom å nyttiggjøre seg av objektstrukturen. Vi har valgt å ikke gjøre dette nettopp fordi programmet baserer seg på kontinuerlig oppdatering av studentinformasjon, hvilket betyr at filene raskt vil være utdaterte. Dog ser vi at det eksempelvis ved semesterslutt, og ved deling/integrering av informasjon til andre systemer, vil kunne være en nyttig funksjon.

Særdeles kort oppsummering

I denne rapporten har vi redegjort for framdriftsplanen vi jobbet etter gjennom dette arbeidsprosessen. Vi har redegjort for planleggingsprosessen og de utfordringer vi møtte på her, samt utforming av programmet og vurdering av pseudokode opp mot ferdig program.

Videre vil det nå følge refleksjoner fra hvert gruppemedlem.

Studentrefleksjoner (½ - 1 side)

Anders:

I denne løsningen har jeg bidratt men Klassen Oppgave. Klassens oppgave er å teste om studentene har fått sine oppgaver og arbeidskrav godkjent, og følgelig «godkjenner» om de kan gå opp til eksamen.

Utviklingen av klassen gikk forholdsvis lett. Bruk av pseudokode var til god hjelp. Tidligere i kurset har jeg gått rett på problemløsningen i utviklingsverktøyet. Jeg ser i ettertid at bruk av pseudokode er et fornuftig startsted når du skal angripe et problem.

Tekniske utfordringer

Noe jeg tar med meg videre fra dette prosjektet er de erfaringene jeg har hatt rundt oppdateringer og utveksling av filer. Jeg synes det har vært vanskeligere å samarbeide om kode. Et prosjekt som dette krever mye testing og små endringer fortløpende. Og skal du få hele programmet til å kjøre krever det at du sitter med siste oppdaterte versjonen fra alle i gruppa. Det finnes enklere måte å dele kode på enn slik vi har gjort det. Dersom jeg skal gjøre et slikt prosjekt en gang til vil jeg foreslå for gruppen at vi lager et felles repository i Eclispe, slik at alle jobber mot oppdaterte versjoner av filene. Et felles repository gjør det lettere å distribuere og få tilgang til oppdatere filer.

Resultat/Egen læring

Jeg synes gruppen kan si seg fornøyd med resultatet. Vi har fått en løsning som dekker kravene oppgaven stiller. Og vi har en løsning som kjører uten feil. Dette har vært en krevende oppgave og alle har lagt ned mye arbeid i løsningen. Ja, hele kurset har vært intenst og krevende, men jeg kan si at jeg har lært mye denne høsten.

Anne Line:

Gjennom arbeidet med denne gruppeoppgaven har jeg bidratt med å lage klassen Start og Main. Klassen Start er en veldig liten klasse som kun har som oppgave å hente inn tidligere objekter og å starte hele programmet. Klassen Main er en noe større klasse som gir brukeren valgmuligheter for deretter å sette i gang metoder som samler inn data, enten fra brukeren eller tidligere objekter.

Gruppen har for det meste samarbeidet over nett, ved å bruke ulike teknologier for både video, lyd og skjermdeling. Dette har vært veldig nyttig for meg, for da har jeg fått mulighet til å komme med spørsmål rundt ting som er uklart eller som jeg rett og slett har oversett. Vi har også hatt et tett samarbeid på dager der vi ikke har hatt nettmøter, og alle har svart raskt på henvendelser fra diverse gruppemedlemmer. Personlig liker jeg best å samarbeide face-to-face, men når dette ikke er mulig er videokonferanser et ekstremt nyttig verktøy, særlig under diskusjon av oppgave og hvordan man går frem når man koder.

Under arbeidet med å lage dette programmet har det oppstått noen irritasjonsmomenter, men også en del "aha"-opplevelser ("er det sånn det funker, ja"). For det første var det veldig uvant å ikke være delaktig i alle klassene som ble opprettet, rett og slett fordi jeg aldri helt fikk følelsen av hva programmet skulle gjøre som helhet. Dette er nok helt klart en vanesak, men jeg tror nok også at vi kunne ha hatt en bedre pseudokode i bunn for å ha noe å støtte oss til når vi ble usikre på hva som faktisk foregikk. Personlig så brukte jeg en del tid på å sette meg inn i hva som foregikk og hvilke typer av data som ble sendt hvor, før jeg overhodet klarte å kode Main klassen. Dette førte til at jeg først satt og kodet slik jeg ville gjort det hvis det hadde vært en individuell oppgave, ved å opprette og lage de andre klassene også. Dette tok selvsagt ganske så lang tid, men gjorde også at jeg fikk en annen type forståelse for hvordan min klasse skulle fungere i samspill med de andre sine klasser.

Noe av det jeg brukte mest tid på var å fikse slik at klassen sjekker alt av input fra brukeren og sørge for at programmet ikke avslutter på en feil måte. Her la jeg inn diverse tester på om brukeren trykker *kryss* eller *cancel* og også om brukeren lot inputfelt stå tomt. I tillegg er det testere på om input er riktig i forhold til den informasjonen vi ønsker. Jeg brukte lang tid på dette fordi jeg ikke ville at brukeren skulle ha mulighet til å kræsje programmet, men få en ny mulighet hvis den hadde bommet på noe.

I tillegg hadde jeg et stort ønske om å klare å lage programmet vindusbasert, men der strakk rett og slett ikke min kunnskap til og jeg måtte gå for noe som var innenfor mitt kunnskapsområde. Derfor kan vi se at programmet noen steder hopper mellom å være dialogboksbasert og å være vindusbasert, siden Øystein hadde laget noen vindusbaserte klasser som blir brukt.

Ida:

Gjennom denne oppgaven har jeg i aller høyeste grad fått kjent på egen usikkerhet, identifisert kunnskapshull og innsett hvor stort og omfattende javaspråket egentlig er. Det skal sies at mitt ambisiøse vesen fikk kjenne på arbeidsmengden gjennom året, med både praksis og tilleggsfag i tillegg til dette kurset. Likevel skal det sies at læringsutbyttet har vært enormt, og det faktum at kurset og denne oppgaven har vært kjempespennende, ofte har resultert i at det er blitt prioritert.

Jeg ønsket ansvar for klassen GenererGruppe, da jeg oppfattet denne klassen som variert og spennende, med mange ulike momenter og metoder. Før jeg kunne starte på selve programmeringen var det for meg nødvendig med en forfining av pseudokode. Dette både fordi jeg selv trengte å skissere klassens metoder i detalj, men også for å få bekreftet fra gruppen at min oppfatning av klassen stemte overens med deres. Da dette var gjort kodet jeg et førsteutkast av samtlige klassen basert på det jeg visste om de andre klassene og deres oppgaver. I denne prosessen ble logikken min testet kontinuerlig, og det viste seg tidlig at jeg har et enormt behov for å kontekstualisere de ulike kodebitene; spesielt i arbeidet med array. Her var det godt å ha kompetansen til de andre gruppemedlemmene i bakhånd, og de delte gledelig sine forslag til endringer i den uendelig lange listen feilmeldinger. Etter hvert som forståelsen for de andre klassene økte, og da spesielt student-klassen, student-objektene og arrayen med disse objektene, gikk arbeidet med klassen greiere.

De mest utfordrende metodene å kode var de som benyttet seg av sortering av array, og her var det en del søk på nett, sammen med kontinuerlige tilbakemeldinger fra gruppen, nødvendige tiltak. Øystein hadde her et par fiffige løsningsforslag jeg ikke kjente til tidligere, men gjennom å gjøre meg kjent med funksjonen til de ulike kodebitene oppfatter jeg disse løsningene som oversiktlige og særdeles lesbare; selv om det ikke nødvendigvis er kode vi har blitt kjent med gjennom selve kursets innhold.

Jeg har opplevd arbeidet i gruppen som særdeles effektivt. Det vi absolutt kunne løst bedre var deling av kode. Her sendte vi ulike versjoner i hytt og gevær, mens vi kanskje heller skulle delt en mappe på eksempelvis Dropbox, hvor vi kunne erstatte gjeldende fil med oppdatert versjon kontinuerlig. På denne måten ville hele gruppen hele tiden hatt tilgang på nyeste versjon, og hvert gruppemedlem kun ansvar for å holde styr på egen klassen. Dette er noe som har blitt klart for meg i avslutningsfasen, hvor det var en del ulike versjoner å holde styr på; noen kjørbare, andre ikke.

Øystein:

Vi har benyttet tjenesten Appear.in, Facebook, Adobe Connect og OneDrive til kommunikasjon, noe som fungerte utmerket. Første samlingen ble vi enige om en felles pseudokode for innlevering. Oppgaven bød på noen utfordringer ettersom de fleste naturlignok tenkte at deres egen kode var lettest/best egnet. Det er ingen faste regler ved Pseudokode, så at den enkelte deltager syntes sitt eget system muligens var enklest kom knapt som en overraskelse.

Likevel etter litt diskusjoner ble gruppen enige om å gå for min pseudokode som et utgangspunkt. Avgjørelsen ble, slik jeg tolket det, tatt på grunnlaget at koden var tydelig delt inn i klasser. Faktumet at det kun er jeg som har tidligere erfaring med javaprogrammering spilte nok også en sterk rolle.

Vi finpusset litt på pseudokoden og la til noen endringer gruppa ble enige om. Dokumentet ble levert som en gruppeinnlevering i Fronter innen tidsfristen. Tilbakemelding fra lærer poengterer at det må forklares mye mer, noe jeg selvfølgelig ser i ettertid. Innleveringen ble forbedret med ytterligere forfining og forklaring av klasser og metoder.

Klassene fordelte vi mellom oss ved hjelp av å melde ønske om klasse på Fronter. Vi er tydeligvis en hensynsfull gruppe ettersom ingen ønsker en klasse som andre hadde valgt. Dette gikk med andre ord smertefritt.

I løpet av oppgaven, så har jeg prøvd å være bevisst på forskjellen i javaerfaring og hjelpe andre i gruppa så godt jeg kan uten å overstyre/endre koden. I slutten av november kastet deltageren Vu inn håndkle, og en av programmets større klasser ble sittende uten forfatter. Jeg tilbød meg å overta ansvaret for denne klassen.

Alt i alt synes jeg dette har vært en god erfaring. Pseudokode er absolutt nytt for meg og det samme gjelder samarbeid om kode. Det har nok kommet et par ekstra små grå hår på hodet under oppsettet av GIT-hub. I tillegg ser jeg fordelen med en pseudokode; det lar en dele programmeringsbitene opp i mindre, mer forståelige, biter. Den forfinede pseudokoden gjorde samarbeidet noe enklere ettersom vi kunne se hvilke navn og funksjoner det kom til å bli mulig å kalle fra andre klasser. Dette kommer til å bli brukt i fremtidige prosjekter.

Avslutningsvis må det innrømmes at det til slutt ble et ganske stort avvik mellom pseudokode og endelig produkt. Nærme innleveringsfrist gikk det opp for oss at vi må gi brukeren mulighet til å redigere studenter og oppgaver. Løsningen ble å implementere en del hastekodede klasser for dette. Disse klassene (Dialog og flere GUIxxxx) var ikke planlagt i pseudokoden. Dette kunne muligens vært unngått dersom vi hadde tegnet et relasjonsdiagram slik at vi hadde fått visualisert samarbeidet mellom klassene internt og brukergrensesnittet.