UNIVERSITETET I OSLO
Institutt for informatikk

Administrasjon og vurdering av programmings-besvarelser

En prototypeimplementasjon av et programvarerammeverk for automatisk og manuell evaluering og skåring av Java oppgaver

Masteroppgave
(60 studiepoeng)

Arne Alexander Salicath

8. juni 2008

Abstrakt: Å få gode indikatorer på prestasjons- og ferdighetsnivåer blant programvareutviklere har mange nyttige anvendelser, blant annet i rekrutteringsøyemed, opplæring og ressursplanlegging i IT-prosjekter. Slike indikatorer kan også brukes for å få en mest mulig balansert allokering av forsøkspersoner (vha blokking) i kontrollerte software engineering eksperimenter, slik at disse får økt validitet. I de senere år er det blitt stadig vanligere å benytte realistiske programmeringsoppgaver som krever at studenter eller forsøkspersoner må skrive programkode. Den enkelte oppgave kan i etterkant vurderes med forskjellige vurderingsmetoder (funksjonell korrekthet, tid, kodemetrikk, lesbarhet, dokumentasjon, vedlikeholdbarhet osv.) eller skåringsregler (for eksempel vektlegging av tidsforbruk versus kvalitet på løsning). I tillegg kan den enkelte oppgave være av forskjellig lengde og struktur, for eksempel oppgaver bestående av flere del-innleveringer med korte tidsfrister eller større oppgaver med mer fleksible tidsfrister. Valg av vurderingsmetode, skåringsregel og oppgavestruktur avhenger av mange faktorer; for eksempel tilgjengelig maskinell støtte for vurderingsmetoder, formålet med undersøkelsen eller tilgjengelighet av manuelle karaktersettere. Uansett hvilken spesifikk metode, skåringsregel eller oppgavestruktur som velges, vil man behøve en stor mengde besvarelser for å få gode mål på prestasjons- og ferdighetsnivå, noe som medfører at det trengs programvare for å effektivt kunne administrere og vurdere oppgavene i etterkant.

Problemstillingen for denne oppgaven er hvordan bygge et rammeverk slik at nye metoder for vurdering kan legges til på en fleksibel måte, og slik at skåringsregler kan forandres. I tillegg må det eksistere muligheter for å sammenligne forskjellige karaktersettere, som igjen benytter seg av forskjellige vurderingsmetoder og skåringsregler. Rammeverket må også kunne administrere oppgavene slik at nye vurderinger basert på nye vurderingsmetoder, endrete skåringsregler eller nye karaktersettere kan gjennomføres på en effektiv måte.

Målsetningen for denne oppgaven er først å utvide funksjonaliteten til en eksisterende prototyperammeverk som er under utvikling ved Simula Research Laboratory. Dernest, å utvikle funksjonalitet for administrering av eksisterende data fra et gjennomført eksperiment hvor det stilles forskjellige krav til nye vurderingsmetoder, skåringsregler og karaktersettere.

Takk til:

Denne oppgaven er skrevet for Institutt for Informatikk ved Universitetet i Oslo.
Jeg vil først og fremst takke min hovedveileder, Gunnar Bergersen, for å ha vært tålmodig, og for å ha pushet på meg når det har vært nødvendig. Jeg vil også rette en takk til Erik Arisholm som har kommet med innspill, blant annet med henblikk på struktur. Til slutt vil jeg takke familie og venner.

Arne Alexander Salicath
Oslo, mai/juni 2008

Innhold

TAKK TIL: 4

INNHOLD 5

1. INTRODUKSJON....11

1.1 GENERELLE PROBLEMSTILLINGER....12

1.2 SPESIFIKK ANVENDELSE....14

1.3 PROBLEMSTILLING....14

1.4 MÅLSETTINGER / BIDRAG....17

1.5 VIDERE STRUKTUR....18

2. RELATERTE ARBEIDER....19

2.1 TESTTEORIER: KLASSISK TESTTEORI (KTT) OG ITEM RESPONS TEORI (IRT)....21

2.2 EKSISTERENDE RAMMEVERK....21

2.3 KONSEPTER....24

2.3.1 Besvarelser....24

2.3.2 Vurderingsmetoder....24

2.3.3 Test og eksperimenter....25

2.3.4 Item....26

2.3.5 Item bank....27

2.3.6 Computer-adaptive tester....27

2.3.7 Programmeringsferdigheter versus prestasjonsnivå....28

2.3.8 Prestasjon og effektivitet....28

2.3.9 Regresjonstesting, funksjonalitet og tidsforbruk/effort (RFE)....29

2.4 ANDRE BEGREPER....29

2.4.1 Brukere *29*

2.4.2 Managers *30*

2.5 MER OM COMPUTER-ADAPTIV TESTING 30

2.5.1 Item utvelgelse og item bank kalibreringer *30*

2.6 BEGRENSNINGER I EKSISTERENDE RAMMVERK 31

3. UTVIKLINGSPROSESSER OG METODER 33

3.1 TESTMETODER 33

3.2 TESTDREVET UTVIKLING 33

3.3 SCRUM 33

3.4 SMART DOKUMENTER 34

3.5 VERKTØY 34

4. ANALYSE AV PROBLEMDOMENET OG KRAVSPESIFIKASJON 35

4.1 ADMINISTRERINGSPROSESSER I ET RAMMEVERK: 35

4.2 JCAT: UTGANGSPUNKTET FOR OPPGAVEN 36

4.3 GENERELL BESKRIVELSE AV FUNKSJONELLE KRAV 36

4.4 ADMINISTRASJON 37

4.4.1 Innhenting av besvarelser *37*

4.4.2 Organisering av besvarelser *38*

4.4.3 Tidsforbruk *38*

4.4.4 Sammenligning av skåringer *39*

4.5 EVALUERING OG SKÅRING AV BESVARELSER 40

4.5.1 Delsvar *41*

4.5.2 Regelsett *42*

4.5.3 Håndtering av menneskelig skåring *42*

4.5.4 Uavhengighet 43

4.5.5 Utvidelse av vurderingsmetoder 44

4.6 IKKE-FUNKSJONELLE KRAV 45

5. DESIGN OG IMPLEMENTASJON 47

5.1 ADMINISTRASJON 47

5.1.1 Innhenting og organisering 47

5.1.2 Tidsforbruk: 50

5.1.3 Rapportering av skår gitt av ulike karaktersettere 51

5.2 EVALUERING OG SKÅRING AV BESVARELSER 52

5.2.1 Implementering av evaluering av pakkede besvarelser 52

5.2.2 Implementering av Graybox evaluering 53

5.2.3 Implementering av enhetstestevaluering 55

5.2.4 Differanse evaluering 57

5.2.5 Vurderinger av delsvar 58

5.2.6 Tidsforbruk for besvarelser av hel- og deloppgaver 60

5.2.7 Implementering av regelsett for skåring 61

5.2.8 Karaktersettere 62

5.3 ENDRET ARKITEKTUR FOR RAMMEVERKET JCAT. 63

6. DISKUSJON 67

6.1 VALG AV VERKTØY 67

6.2 ANALYSE AV RESULTATER 67

6.2.1 Administrering 67

6.2.2 Evaluering og skåring av besvarelser 71

6.3 FORBEREDELSE FOR COMPUTER-ADAPTIV TEST 73

6.4 IKKE-FUNKSJONELLE KRAV 75

6.4.1 Plattformuavhengighet 75

6.4.2 Åpen kildekode (Open Source) 75

6.4.3 Andre ikke-funksjonelle krav 76

6.5 ERFARINGER 77

6.5.1 Prosjektarbeid 77

6.5.2 Eget arbeide 78

7. KONKLUSJON OG VIDERE ARBEIDE 81

7.1 ADMINISTRASJON 81

7.1.1 Innlastingen av besvarelser 81

7.1.2 Innhenting av data fra front end 82

7.2 VURDERING AV BESVARELSER 83

7.2.1 Evalueringsmetoder 83

7.2.2 Skåring 83

7.3 FORBEREDELSE TIL COMPUTER-ADAPTIVE TESTER 84

7.4 VIDERE ARBEIDE 85

ORDLISTE 87

KILDELISTE 89

VEDLEGG 91

VEDLEGG A – DELSVARSHÅNDTERINGER 91

VEDLEGG B – INNHENTING AV MANUELT SATTE KARAKTERER/SKÅR 92

VEDLEGG C – LISENSER 93

FIGURER:

Figur 1 Progresjon i testlet. 27

Figur 2 SCRUM utviklingsproses. 34

Figur 3 Rammeverkets plassering i forhold til interesseområde. 35

Figur 4 Ulike måter å vurdere tidsforbruk..... 38

Figur 5 Ulike formater på skåringsrapporter. 39

Figur 7 Plassering av et grensesnitt ”Graybox” i forhold til testcase og besvarelse. 43

Figur 8 Feil i skår forårsaket av tidligere besvarelser 44

Fig. 9 Bruksmønstre for dataimportering. 47

Fig. 10 Struktur mottatt fra front end. 48

Fig. 11 Ny struktur etter importering til katalogstruktur. 49

Fig. 12 Tidsforbruk for srcA i figur 9 51

Figur 13 Skjematisk plassering av Graybox mellom testcase og besvarelse..... 53

Figur 14 Kall til metoder gjennom Graybox. 54

Fig. 15 Hendelsesforløp for skåring av delsvar. 59

Fig. 16 Tidsforbruk for deloppgaver. 60

Fig. 17 Komponentbeskrivelse av rammeverket JCAT..... 64

TABELLER:

Tabell 1 Oversikt over rammeverk og egenskaper ved disse 22

Tabell 2 Utdrag av besvarelser lagret i database 49

Tabell 3 Kobling av data mellom SESE og JCAT 51

Tabell 4 Tabellformat for en standard karaktersetter**.** 52

Tabell 5 Tabellformat med karaktersetter 52

Tabell 6 Plassering av tidsforbruk i database 60

Tabell 7 Regler for skår på tabellform 62

1. Introduksjon

De siste årene har man sett at bruk av dataassisterte vurdering av programmeringsbesvarelser og av dataassistert læring innen dataprogrammering tas oftere i bruk. Ikke bare gjelder dette utdanningsinstitusjoner, men man ser også at det tas i bruk i programmeringsindustri og i programmeringskonkurranser (Ala Mutka 2005). Spesielt har programvarerammeverk som Online Judge (Chang, Kurnia et al 2003), HoGG (Morris 2003) og andre bidratt til dette gjennom effektiv vurdering av de besvarelser som leveres inn. Fordi dataassistert vurdering av programmeringsbesvarelser kan utføres på en effektiv og konsistent måte, brukes det i økende grad (Denise M. Woit & Mason 1998; Denise Woit & Mason 2003; Edwards 2003; 2003). Disse egenskapene gjør at slike vurderingsmetoder også egner seg i konkurranse og rekrutteringssammenhenger (Denise M. Woit & Mason 1998; Duane Buck & Stucki 2001).

Motivasjonen for å bruke dataassistert vurdering av innsamlede programmeringsbesvarelser, er at man effektivt og konsistent kan vurdere programmeringsbesvarelser for å måle prestasjons- og ferdighetsnivåer. Gjennom å vurdere besvarelser fra flere subjekter på ulike oppgaver på en tidsbesparende og systematisk måte, kan man se hvordan disse subjektene presterer i forhold til andre subjekter. For å kunne teste ulike egenskaper ved store mengder besvarelser, bør disse besvarelsene administreres slik at man enklest kan hente frem besvarelsene for effektiv vurdering. Man trenger også å organisere enkelte delsvar, slik at de kan vurderes enkeltvis eller sett som en del av en større oppgave.

I forbindelse med organisering og testing av programmeringsbesvarelser finnes det flere typer programvarerammeverk. Noen av disse benyttes til å sende ut oppgaver og motta besvarelser for deltakere i tester, men i tillegg kan man ha egne rammeverk som håndterer den direkte kommunikasjonen mot bruker og/eller subjekt. Slike rammeverk kalles ”front end" fordi de tilbyr et grensesnitt mot ulike typer sluttbrukere. Det benyttes i noen grad også egne rammeverk som har til oppgave å utføre vurderinger av besvarelser som mottas. Generelt for flere slike programvarerammeverk, er at de må håndtere store mengder besvarelser. For at man skal kunne vurdere besvarelsene automatisk eller semi-automatisk, må rammeverk som skal utføre slike vurderinger være i stand til å motta besvarelser. Disse inngår som del av tester[1] på programmeringsferdigheter. Begrepet ”test” er definert som en systematisk prosedyre for å observere

[1] Test: ”Eksperiment, prøve, metode i psykologien karakterisert ved en standardsituasjon som forskjellige individer kan stilles overfor, og hvor man er interessert i de individuelle forskjeller i prestasjoner og reaksjoner[…]”. Kilde: Aschehoug og Gyldendals Store Norske Leksikon, 3. utgave, 2.opplag, 1999.

og beskrive kunnskap, evner eller ferdigheter ved bruk av kvantifiserbare størrelser (Cronbach 1990). To eksempler på formål en test kan være utviklet for, kan være i hvilken grad en student har lært seg sentrale deler av et kursmateriale, eller hvor gode ferdighetene er til en spesiell programvareutvikler er.

Alle besvarelser en person leverer i løpet av en test blir vurdert i etterkant og det er vanlig å skille mellom tester som vurderes online og offline. Forskjellen mellom disse er at online vurderinger skjer samtidig som besvarelser mottas slik at resultatet for en spesifikk oppgave er tilgjengelig i løpet av få sekunder. I motsetning til online vurderinger, vil offline vurderinger skje lang tid i etterkant av at besvarelsene er mottatt og testen er ferdig utført. Denne oppgaven vil kun se på sistnevnte form for vurderinger. Videre kan også disse formene for vurderinger deles opp i både statiske[2] og dynamiske[3] vurderinger (vurderinger av kjørende kode). Begge disse formene er aktuelle i denne oppgaven, men de statiske vurderingene utføres kun manuelt eller semi-automatisk i denne oppgaven.

1.1 Generelle problemstillinger

Administrasjon kan deles inn i to underområder: Det første underområdet er innhenting av data, og det andre området er hvordan disse skal organiseres mest effektivt. Innhentingen av data gjøres gjennom å benytte en ”front end” mot subjektene for å sende ut oppgaver og motta besvarelser. Disse bør igjen organiseres slik at vurderingene kan skje mest mulig effektivt og korrekt. Dermed kan besvarelser for oppgaver med en gitt vanskelighetsgrad sammenlignes. ”Karakterer” settes på bakgrunn av hvordan besvarelsen oppfyller en rekke krav til resultater som man forventer den skal produsere, og hvor disse kravene vektes i forhold til hverandre. En ”karaktersetter” er videre en person eller en algoritme som gjennomfører vurderinger av en oppgave og deretter setter en karakter. Fordi eksisterende vurderinger ikke tar høyde for alle måter å vurdere oppgaver på, trenger man mulighet for å legge til nye. Disse vurderingsmetodene trenger man å legge til på en slik måte at man unngår å måtte gjøre endringer i de eksisterende vurderingsmetoder. Man trenger også å kunne vurdere besvarelser opp mot ulike kriterier for å se hvordan besvarelsene skårer på ulike områder. En utfordring er altså å lage et rammeverk som er utvidbart i forhold til nye vurderingsmetoder. Mer spesifikt, vurderingsmetodene består i å evaluere[4] og skåre[5] besvarelsene.

---

[2] Statisk vurdering: Vurdering gjort på et program ikke kjøres. (Ala Mutka 2005)

[3] Dynamisk vurdering: Vurdering gjort på kode som kjører. (Ala Mutka 2005)

[4] Evaluering: En kvalitativ bedømmelse. Kilde: Aschehoug og Gyldendals Store Norske Leksikon, 3. utgave, 2.opplag, 1999.

Det finnes flere måter å vurdere besvarelser på. I de fleste tilfeller er man interessert i å vurdere korrektheten av besvarelsen, det vil si at den produserer de resultater man forventer den gjør. Men i andre sammenhenger kan det være andre egenskaper som er like interessante. Fordi det kan være ønskelig å teste slike egenskaper, burde slike kriterier kunne legges til, uten å komme i konflikt med eksisterende metoder. Ved å legge til nye metoder, kan man også studere påvirkninger dette får på resultatet. Metodene som nevnes her er automatiske vurderingsmetoder, men det finnes eksempler hvor automatikk ikke kan benyttes. Når lesbarhet og gjenbrukbarhet skal vurderes, kan ikke automatikk utføres (Cheang, Kurnia et al. 2003). For å effektivt finne forskjeller mellom ulike besvarelser, kan man kombinere automatiske og manuelle vurderingsformer. En kombinasjon av disse to vurderingsformene kan illustreres med differansevurderinger, hvor man først utfører generering av forskjellsrapporter automatisk og deretter vurderer de enkelte rapportene manuelt i etterkant. Man har også flere alternativer for å vektlegge de enkelte kriteriene for oppgaver. Enkelte oppgaver kan vektlegge funksjonalitet, mens andre igjen kan vektlegge andre kriterier som bruk av gode abstraksjoner, god dokumentasjon og gjenbrukbarhet. I denne sammenhengen er man kun interessert i de svar som går på konkrete programmeringsoppgaver, og som resulterer i målbare verdier.

En annen utfordring er å lage et rammeverk som er fleksibelt og utvidbart med hensyn til hvilke oppgavetyper det kan håndtere. I noen oppgavetyper ønsker man kun enkle svar, mens i andre oppgavetyper kan det være nødvendig med mer sammensatte svar. Med enkle svar menes svar som tar utgangspunkt i elementære enheter mens sammensatte besvarelser kan inkludere flere enheter som inngår i besvarelsen. I programmeringsrelaterte oppgaver finnes det flere typer besvarelser å forholde seg til, noe som igjen gjør at man trenger å ha utvidbarhet på dette området når flere typer skal legges til. Deloppgaver skiller seg fra vanlige oppgaver fordi de har flere spørsmål som hver trenger sine besvarelser. Vurderinger av de enkelte delsvar inngår i en totalvurdering av oppgavebesvarelsen. Utfordringen er å kunne administrere og vurdere delsvarene individuelt, samtidig som evalueringene kan inngå i skåringen av hele besvarelsen.

[5] Skår (score): ”Antall poeng som er oppnådd ved f.eks. en test”. Kilde: Aschehoug og Gyldendals Store Norske Leksikon, 3. utgave, 2.opplag, 1999.

1.2 Spesifikk anvendelse

Som del av et større forskningsprosjekt ved SRL, lages en pilot for computer-adaptive tester[6] i rammeverket ”Java programming Computerized Assessment/Adaptive Test” (videre forkortet JCAT). Dette rammeverket skal brukes for testing av subjekters programmeringsferdigheter, og er laget som del av doktorgradsavhandlingen til Gunnar Bergersen. Ved å ta utgangspunkt i dette rammeverket, ser jeg på hvordan større mengder besvarelser kan importeres og organiseres som en del av administreringen av disse, slik at man effektivt kan vurdere besvarelser hentet fra et pågående forskningsarbeid ved Simula. Gjennom å tilby informasjon om resultater for ulike oppgaver, kan dette rammeverket også klargjøres for computer-adaptiv test, slik at mer reelle ferdighetsestimater kan dannes. For å definere ulike eksperimenter benyttes rammeverket SESE - Simula Experiment Support Environment (Arisholm, Sjøberg et al. 2002) som front end. Front end er programvare som kommuniserer direkte med bruker. Gjennom dette rammeverket kan ulike tester og eksperimenter defineres og administreres, og progresjoner kan overvåkes. Disse testene og eksperimentene kan inneholde et rikt utvalg av oppgaver som kan leveres til større mengder forsøkspersoner (subjekter). I dag håndterer SESE datainnsamling fra subjekter med henblikk på å teste ferdighetsnivå, slik at besvarelsene kan vurderes i etterkant ved hjelp av JCAT.

I adaptiv testing bestemmes ”neste oppgave” i testen av resultatet fra foregående oppgave. Dette forutsetter at vanskelighetsgraden på oppgavene er kjent. Adaptive tester forutsetter kalibrerte oppgaver med hensyn på vanskelighetsgrad. For å få målbare verdier på vanskelighetsgrad, er det en fordel med flere ulike karaktersettere. En karaktersetter bruker resultatene fra en eller flere evalueringsmetoder og gir en aggregert skår basert på en skåringsregel. Ved å bruke data fra flere karaktersettere får man et bedre grunnlag for å estimere vanskelighetsgraden på oppgavene.

1.3 Problemstilling

Administrering av besvarelser medfører følgende problemstillinger knyttet til innhenting og organisering av besvarelser:

- Innhenting av besvarelser og tidsforbruk skjer gjennom å benytte seg av SESE.

[6] Computer-adaptiv test: En test, eksperiment hvor rekkefølgen på oppgaver bestemmes gjennom dataassistert vurdering av besvarelser.

- Besvarelsene som mottas fra SESE kan komme i grupper med ulik antall besvarelser som inngår i hver gruppe. Man kan hente ut alle besvarelsene, eller man kan velge å hente ut et utvalg av disse. I tillegg har man organisert dataene slik at de ligger i henhold til oppgavene eller deloppgavene det gjelder, men man ønsker gjerne å knytte det sammen i henhold til subjektet som leverer oppgaven.

- Tid kan også være en avgjørende faktor for hvordan vurderinger av besvarelser skal utføres. I de fleste oppgaver defineres gjerne krav til tidsforbruk. Utfordringen er å finne måter å knytte informasjon om dette til den konkrete besvarelsen, slik at denne informasjonen kan inngå i skåring av oppgavebesvarelsen. Tidsinformasjonen, som ble innhentet av SESE, kan også kunne knyttes opp mot de besvarelser som organiseres gjennom JCAT dersom den skal inngå i vurderingskriteriene for oppgavene. Fordi man kan snakke om store mengder besvarelser, vil mengden av antall besvarelser gi også utfordringer i forhold til hastighet, både på innhenting og gjennomføringer av vurderinger.

- Data hentet fra SESE trenger også å organiseres i JCAT for at vurderinger av besvarelser skal kunne utføres automatisk eller semi-automatisk. En av utfordringene i forhold til denne organiseringen, er hvorvidt besvarelser lagres persistent i en database eller katalogstruktur. Dette er en utfordring fordi man på den ene side lett kan få tilgang til besvarelser gjennom en database, men på den annen side ikke kan åpne innholdet direkte uten først å gå veien om katalog og filstruktur. Dersom de allerede ligger på filstruktur, slipper man dette problemet, men samtidig mister man en del muligheter som man har gjennom å benytte en database. Andre utfordringer er å konvertere strukturen som er hentet fra SESE over på en struktur som egner seg bedre til å utføre de vurderinger som trengs.

- For å kunne klargjøre for adaptive tester, er det flere utfordringer som må løses. For det første trenger man å ha gode data for å kunne kalibrere oppgaver. Videre trenger man gode måter å representere dataene på, slik at kalibreringen[7] lettest mulig kan utføres. Dernest trenger man å vite at resultater av de ulike metoder ikke inneholder feil som kan påvirke utfallet av testene. Til slutt har man utfordringer i å finne frem til gode utvalgsmekanismer som kan brukes i utvelgelsen av oppgaver. Metoder for kalibreringer og utvelgelser av oppgaver vil kun bli overfladisk behandlet i denne oppgaven.

[7] Kalibrering: Fastlegging av skala på et måleinstrument og angivelse av fysisk størrelse. Kilde: Aschehougs og Gyldendals Store Norske Leksikon, 3.utgave, 2. opplag. Oslo, 1999. Se også ordliste.

- For å se hvordan ulike karaktersettere vurderer besvarelsene trenger man sammenlignbare resultater fra disse. Man har også behov for tilsvarende informasjon for å kunne kalibrere de ulike oppgavene i forbindelse med computer-adaptiv test.

- Fordi man ikke vet om vurderingsmetodene man introduserer inneholder feil, er det en fordel at de etterkontrolleres. Ved å gjøre sammenligninger med kjente besvarelser, kan man sammenligne resultatet av de nye vurderingsmetodene med kjente verdier. Ut fra denne sammenligningen kan man, på bakgrunn av likheter og eventuelle forskjeller, komme frem til konklusjoner om hvorvidt metodene kan inneholde feil.

Vurderinger: For å vurdere nivået på de ulike besvarelser, finnes det flere ulike evalueringsmetoder og skåringsregler som kan benyttes. Noen egenskaper ved programkoden kan testes automatisk mens andre kan kun testes manuelt.

- Manuell karaktersetting dannes på bakgrunn av vurderinger som gjøres av menneskelige ressurser. For at disse vurderingene skal inngå i datagrunnlaget for å estimere subjekters ferdighetsnivå, bør de behandles på lik linje med karakterer som settes automatisk. I den forbindelse gjøres det implisitt evaluering av besvarelser, slik at man ikke skiller mellom manuell evaluering og karaktersetting. Programmeringsbesvarelser som går inn i en uendelig løkke vil også kreve manuell karaktersetting, samt for å vurdere lesbarhet og gjenbrukbarhet. Hvordan kan data fra karaktersettere håndteres, og hvordan kan man håndtere ulike karaktersettere? Selv om slike karaktersettinger er subjektive har de allikevel verdi. Spesielt interessant er dette i forbindelse med vurdering av oppgavebesvarelser hvor automatiske vurderingsmetoder ikke kan benyttes, slik som vurdering av brukervennlighet eller lesbarhet.

- Når man opererer med automatisk karaktersetting trenger man å skille karaktersetting fra evaluering fordi man på et senere tidspunkt kan erstatte evalueringsmetodene. Man trenger i forbindelse med automatisk karaktersettere å skille evalueringskonseptet fra skåringskonseptet fordi automatiske skåringer kan benytte ulike evalueringer. I noen tilfeller trenger man å definere et sett med skåringsregler som angir sammenhenger mellom ulike evalueringer og karakterer som skal gis.

- For at resultatene av karaktersettinger ikke kan påvirkes av ytre omstendigheter, trenger man å skille mellom testcase (sett av tester som skal utføres) og besvarelsene man ønsker å teste. Utfordringen med dette punktet, er å gjøre det på en slik måte at man kan svartboksteste besvarelser eller komponenter som inngår i besvarelsen. Med svartbokstesting mener man at man tester en struktur uten å skille mellom komponenter som inngår. På grunn av nettopp

denne begrensningen, får man ikke tilgang til å teste disse enkeltkomponentene. Til nå har man har til nå savnet metoder som kombinerer disse egenskapene.

- I tillegg til at det finnes store mengder besvarelser, finnes det også flere ulike typer oppgaver. Enkelte oppgaver har kun ett spørsmål, og behøver kun ett svar. Men i andre oppgaver har man spørsmål hvor man skal levere inn flere besvarelser. Hvordan disse kan innhentes og organiseres, på en slik måte at de ikke kommer i konflikt med andre besvarelsene knyttet til samme oppgave, er også en utfordring som må løses. Man kan også tenke seg at ytterligere andre oppgavetyper kan komme til. Disse burde også kunne legges til på lik linje med andre oppgavetyper.

- Ikke bare trenger man fleksibilitet i forhold til oppgavetyper i slike rammeverk; man trenger også muligheten til å legge til flere vurderingsmetoder for å vurdere besvarelser på bakgrunn av nye eller endrede evalueringsmetodene, uten at man fjerner evalueringer som allerede er definerte. Samtidig er utfordringen å gjøre dette på en måte slik at man ikke behøver å starte alt på nytt.

1.4 Målsettinger / bidrag

Min målsetning ved denne oppgaven er tredelt: å forstå det som skal gjøres med det eksisterende rammeverket, gjennomføre endringene og evaluere endringene i etterkant.

- En kravspesifikasjon skal utarbeides slik at man får en dypere forståelse av hva som trengs for å utvide funksjonaliteten til et programvarerammeverk på følgende områder: Rammeverket skal kunne innhente og organisere større mengder besvarelser av hele eller delvise oppgaver på en effektiv og systematisk måte, slik man kan vurdere besvarelser i henhold til eksisterende og fremtidige kriterier. Vurderingene skal gjøres på en slik måte at de ikke blir påvirket i større grad av ytre faktorer, og resultatene skal kunne sammenlignes med hverandre.

- På bakgrunn av kravspesifikasjonen, skal prototypeløsninger for de problemstillinger nevnt ovenfor designes og implementeres. Med utgangspunkt i data hentet fra et pågående prosjekt ved Simula, vil disse metodene bli kontrollert og validert.

- Løsningene det kommes frem til skal evalueres, og det skal trekkes frem positive og negative sider ved de metoder som er valgt. Disse bør også settes opp mot alternative løsninger. Det skal også gjøres en evaluering av eget arbeid.

1.5 Videre struktur

Kapittel 2 beskriver bakgrunnen for bruken av automatisert og halvautomatisert vurdering av oppgaver, og hvordan disse kontrolleres for korrekthet. Vurderingsmetoder som benyttes i denne oppgaven beskrives i kapittel 3. Kapittel 4 beskriver JCAT samt kravspesifikasjonen og tilhørende analyser. Kapittel 5 beskriver design og implementeringer som er gjort, mens kapittel 6 diskuterer implementeringen og arbeid i prosjekt. Deretter følger en konklusjon og beskrivelser av arbeid som gjenstår i kapittel 7.

2. Relaterte arbeider

Bruk av menneskelig skåring er en av de mest benyttede metoder for å vurdere nivået på besvarelsene, men man har noen utfordringer i forbindelse med disse: 1) Korrekthet og effektivitet er vanskelig for mennesker å vurdere kun gjennom å kjøre en besvarelse, og i programmer som inneholder feil, må årsakene undersøkes. 2) Man kan ha ulike tilnærminger til samme problem, som alle har samme resultat. 3) Menneskelige karaktersettere har tendenser til å legge forholdsvis for stor vekt på vedlikeholdbarhet og lesbarhet. 4) Menneskelige feil kan oppstå, og 5) Menneskelig skåring tar mye tid og er ensformig (Cheang, Kurnia et al. 2003). Menneskelig inkonsistens i skåring og administrasjon av store mengder besvarelser er også utfordring i andre rammeverk, til eksempel HoGG – ”Homework Generation and Grading project” (Morris 2003). I dette prosjektet ser man på hvordan man skal administrere besvarelser fra ca 300 - 600 studenter med 10 - 12 besvarelser per student per semester, noe som er tidkrevende ved bruk av menneskelige ressurser. Selv om antall subjekter er annerledes for denne oppgaven, vil også store mengder besvarelser være en utfordring når rammeverket JCAT skal benyttes med reelle data med henblikk på adaptiv test. Da datamaskiner kan utføre samme metoder repetitivt og konsistent i motsetning til menneskelige ressurser, og kan vurdere kriterier som man ikke kan vurdere manuelt, blir datamaskinassistert vurdering brukt i økende grad. På den annen side er fremdeles vanlig praksis å bruke menneskelig skåring når man skal vurdere egenskaper som man ikke kan vurdere automatisk (Cheang, Kurnia et al. 2003). I enkelte oppgaver har man spørsmål som bygger på tidligere oppgaver. Slike oppgaver gir utfordringer i forbindelse med å gi korrekt skår fordi man får statiske avhengigheter. I klassisk testteorier ser man på den observerte skåren som sammensatt av reell skår pluss en feilskår, men man vet verken den reelle skåren eller hvor stor feilskåren er. Item responsteori derimot, inkluderer betydningen og sammenhengen av ulike item. Samtidig kompenserer den for statistiske avhengigheter mellom item (Luecht, Brumfield et al. 2006).

Noen av begrensningene med enkelte eksisterende rammeverk laget for automatiske vurderinger er at de kun tar høyde for å utføres på lokale maskiner. Mange er også i stor grad tilpasset til de ulike oppgavene de er utviklet for (Ala Mutka 2005), slik at man trenger mer generelle rammeverk. Disse rammeverkene skal vurdere prestasjoner til subjekter, ved først å gi en verdinøytral analyse av besvarelsene. Gjennom å vektlegge ulike egenskaper, kan gi skår på bakgrunn av evalueringene av disse besvarelsene. Når man skal evaluere besvarelsene, er det flere ulike egenskaper og kriterier man ønsker å måle. Den viktigste egenskapen er korrekthet gjennom å måle funksjonalitet, men man ønsker også å sjekke at eksisterende, fungerende funksjonalitet ikke brytes i de tilfeller det er snakk om feilrettingsoppgaver. I tillegg er vedlikeholdbarhet et viktig kriterium, men som det er påpekt tidligere er dette noe som kun kan vurderes manuelt. Noen mener at også effektivitet bør inngå

(Cheang, Kurnia et al. 2003), men man kan diskutere om dette er like viktig med dagens raske maskiner som det var tidligere. I enkelte sammenhenger kan andre egenskaper som robusthet være viktigere. Man snakker om to hovedformer for evaluering av programmeringsbesvarelser: Det ene er statiske evalueringer, hvor man baserer vurderingene på bakgrunn av data som ikke kan endre seg i løpet av tiden evalueringen pågår. Dynamiske vurderinger utføres mens programmet som inngår i besvarelsen kjøres, slik at man kan evaluere data som endrer seg underveis. For at dette skal gjøres på en tilstrekkelig god måte, trenger man å utføre slike vurderinger is trygge omgivelser (Ala Mutka 2005). Trygge omgivelser har ikke samme betydning for statiske evalueringer, som den har for dynamiske fordi programmene ikke trenges å kjøres ved statiske analyser. Statiske evalueringer kan også lett utføres av mennesker fordi dataene ikke endrer seg. Dynamiske evalueringer lar seg vanskeligere gjennomføre ved bruk av menneskelige ressurser fordi men blir vanskeligere å gjennomføre når man skal evaluere besvarelser hvor data kan endre seg.

I noen av forundersøkelsene til denne oppgaven, ble eksisterende rammeverk for å undersøke fellestrekk og ulikheter mellom disse. Felles for alle de eksisterende rammeverkene, bortsett fra JCAT, er at de benytter egne grensesnitt for å motta besvarelser for de oppgaver de administrerer. JCAT mangler eget grensesnitt og trenger å benytte andre front end for å motta besvarelsene. Ved Simula benyttes det et slikt rammeverk som kan i denne sammenheng benyttes som front end. SESE er en applikasjon som tilbyr real time overvåkning av eksperimenter, fleksibel definering av spørsmål med tilhørende måleskala, og benyttes i forskjellige eksperimenter ved Simula (Arisholm, Sjøberg et al. 2002; Arisholm & Chen 2004). De viktigste prosessene den håndterer er å definere oppgaver og de stimuli subjektene skal motta. Den avgrenser også mengden av subjekter som deltar på ulike tester, og den håndterer grensesnittet med mottak av besvarelser. Alle svar som mottas fra de ulike subjektene kan ekstraheres fra SESE, og kan overføres som en eller flere filer. Ytterligere informasjon, slik som tidsforbruk kan også hentes ut. I prosessen rundt innhenting av data i forbindelse med eksperimenter er det fem viktige punkter som inngår (Arisholm, Sjøberg et al. 2002) og som danner grunnlaget for de besvarelser som ønskes å administreres: 1. definisjon av eksperiment, 2. definering, samling og tilordning av subjekter. 3 utføring av oppgaver i eksperimentet. 4. overvåking av eksperimentet, og til sist 5. innsamling av resultater. Det er utdrag av slike resultater man benytter for å vurdere egenskapene til rammeverket JCAT.

2.1 Testteorier: Klassisk testteori (KTT) og Item respons teori (IRT)

Før man ser på eksisterende rammeverk, er det først et par viktige teorier innen testing av besvarelser man først bør se litt nærmere på. Det er to hovedfelter innen testteorier. Den klassiske testteorien postulerer sammenhengen mellom observert skår, reell skår og feil. Den skår man observerer er gitt som summen av den virkelige skåren og feilskår, men utfordringen ligger i at virkelig skår og feilskår er to ukjente variabler. Begrensningen som fremheves med klassisk testteori, er antakelsen om ikke-eksisterende sammenhenger mellom feilskår og reell skår, fordi personer med lengre erfaring gjør færre feil enn en med liten erfaring (Hambleton & Jones 1993). Denne utfordringen går igjen når man skal evaluere og skåre besvarelser av spørsmål, hvor spørsmålene baserer seg på tidligere spørsmål. På grunn av dette, har man introdusert en testteori som kalles Item responsteori. Modellen til klassisk testteori, om at observert skår er basert på reell skår pluss feilskår, er lineær fordi disse anses å være uavhengige. IRT opererer med ikke-lineær modell, hvor man får sammenhenger mellom de ulike variablene. Når man skal utarbeide tester i klassisk testteori er det ni punkter man må uføre. Det første punktet er utarbeidelse av spesifikasjoner for testen. Deretter trenger man å forberede en pool hvor disse oppgavene kan lagres for uthenting, og hver oppgave kalibreres med vanskelighetsgrad. Denne må så pilottestes før man kan utvikle en endelig test. Den endelige testen må så administreres, hvorpå det kan utføres teknisk analyse. Administrative instrukser må deretter forberedes før man kan distribuere ut oppgaver og skrive ut innholdet.

2.2 Eksisterende rammeverk

Før definisjon av konsepter vil jeg først gi en generell oversikt over noen av de ulike rammeverk som ble funnet i forbindelse med prestudiene til denne oppgaven. I forberedelsene til denne oppgaven var jeg interessert i de tekniske aspektene rundt de ulike rammeverkene, men vil her prøve å vise noen av de egenskapene som man kan måle i ulike eksisterende rammeverk, samt hvor hovedvekten av anvendelsen ligger.

Tabell 1 Oversikt over rammeverk og egenskaper ved disse

Navn	Skåring basert på:	Type oppgaver	Anvendelse	Måler:
Assyst[8]	Output		Akademia	Funksjonalitet, effektivitet, testevner, lesbarhet, software metrikk
BOSS [9]	Output		Akademia	Funksjonalitet, automatisk vurdering i innleveringsfase, summativ
CAP [10]	Output	Pascal	Akademia	Programmeringsfeil
Ceilidh [11]	Output og kodestil	Diverse	Akademia	Funksjonalitet, effektivitet, lesbarhet
CourseMarker [12]	Designdiagram, koding, kodingsstil Output	Diverse	Kurs, akademia	Funksjonalitet, formativ
Environment for Learning to Program – ELP[13]	Output	Java	Akademia	Funksjonalitet, software metrikk, strukturell likhetsanalyse
HoGG[14]	Output	Perl og Java	Kurs, Akademia	Tilbyr simulering

[8] (Ala-Mutka 2005)

[9] (Joy, Griffiths et al. 2005)

[10] (Ala-Mutka 2005)

[11] (Ala-Mutka 2005)

[12] (Ala-Mutka 2005)

[13] (Ala-Mutka 2005)

Navn	Skåring basert på:	Type oppgaver	Anvendelse	Måler:
JCAT[15]	Output, metoderesultat	Java	Forskning, Industri	Funksjonalitet, regresjon
Online Judge[16]	Output	C, C++, Java	Akademia	Funksjonalitet, effektivitet, automatisk vurdering i innleveringsfase
Quiver[17]	Analyserer metoder i Java	Java	Akademia	Tilstandspersistens
Scheme-Robo [18]		Pascal, Java	Akademia	Funksjonalitet, design
SESE[19]		Diverse	Forskning	Definering og overvåking av eksperimenter, samt front end
Web-CAT[20]	Output	Pascal, Java o.a.	Akademia	Funksjonalitet
WebToTeach[21]	Sammenligner kodefragment med mal	Diverse	Akademia	Funksjonalitet

[14] Morris, D. S. (2003). Automatic grading of student's programming assignments: an interactive process and suite of programs Frontiers in Education, 2003. FIE 2003. 33rd Annual.

[15] G. Bergersen: Java programme Computerized Assessment/Adaptive Test

[16] B. Cheang, A. Kurnia, et al. (2003). "On automated grading of programming assignments in an academic institution." Computers & Education 41(2): 121-131

[17] Ala-Mutka, K. M. (2005). "A Survey of Automated Assessment Approaches for Programming Assignments." Computer Science Education 15: 83-102.

[18] Som 7.

[19]Arisholm, E., D. I. K. Sjøberg, et al. (2002). "A web-based support environment for software engineering experiments." Nordic J. of Computing 9(3): 231-247., Arisholm, E. og P. M. Chen (2004). "An automated feedback system for computer organization projects." Education, IEEE transaction on 47(2): 232-240.

[20] A. Shah (2003). Web-CAT: A Web-based Center for Automated Testing.

[21]Ala-Mutka, K. M. (2005). "A Survey of Automated Assessment Approaches for Programming Assignments." Computer Science Education 15: 83-102.

Som man ser av tabell 1, er flere av disse rammeverkene knyttet til ulike anvendelsesområder. De er også sterkt knyttet til oppgavetyper de skal teste, noe som også gjelder for rammeverket denne oppgaven tar utgangspunkt i.

2.3 Konsepter

For å gi bedre forståelse av hva de ulike konseptene bidrar med, gis her en oversikt over de viktigste konseptene som benyttes i denne oppgaven.

2.3.1 Besvarelser

En besvarelse av en oppgave er en type respons som skjer på bakgrunn av oppgavespørsmålenes som subjektet mottar. I enkelte situasjoner har man oppgaver som bygger på hverandre og som krever at subjektet leverer inn flere besvarelser, og i denne oppgaven går disse besvarelsene under begrepet delsvar. Besvarelser kan komme i flere former. På enkle spørsmål kan man enten svare med ja/nei, tall eller tekst. Større oppgaver kan igjen kreve mer informasjon, slik som at subjektet skriver et essay, men i denne oppgaven er ikke slike svar aktuelle. Programmeringsbesvarelser er mer aktuelle for denne problemstillingen. Disse svarene er gjerne samlinger av en eller flere programkodefiler. En besvarelsesbank sørger for å organisere innhentingene av besvarelsene.

2.3.2 Vurderingsmetoder.

Evalueringsmetoder: Evaluering er metoder for å kunne gi verdinøytrale målinger, ut fra hvilke kriterier man ønsker å vurdere oppgavebesvarelsene ut i fra. For at evalueringene skal kunne være objektive, kan ikke personlige preferanser inngå. Samtidig kan automatiske evalueringer vurdere besvarelser letter, hvor man får data som endrer seg kontinuerlig. Hensikten i denne sammenhengen er å finne ut hvorvidt besvarelsene oppfyller bestemte krav i oppgaven, hvor mange som oppfylles og hvor mange som ikke er oppfylt. For at målingene skal kunne benyttes i forskningssammenheng, må de ikke påvirkes av subjektivitet. Evalueringskriterier kan i utgangspunktet være objektive eller subjektive. Objektive, målbare verdier som benyttes i denne sammenhengen er antall svar som er korrekte, feil, eller måling av tidsforbruk.

Skåringsmetoder: Skåring av besvarelser benyttes for å vurdere prestasjonene til et subjekt på ulike oppgaver og for å kunne sammenligne denne prestasjonen opp mot andres prestasjoner. I flere sammenhenger har man gjort nytte av flervalgsoppgaver. Slike oppgaver gjør at man lett kan sammenligne besvarelser mot fasit, og dermed gi mer effektiv vurdering fremfor å lese igjennom hele

besvarelsene samt mindre rom for avvik i skåringen. På tilsvarende måter kan besvarelser sammenlignes mot forventede resultater når disse kjøres. Ut fra ulike evalueringene og skåringene som gjøres av besvarelsen, kan man sammenligne hvordan ulike kriterier påvirker skåringene, En karakter dannes på bakgrunn av et sett med kriterier, slik som at man skal ha løst en bestemt andel av oppgavene for å få en gitt karakter, i motsetning til evaluering hvor man kun så på konkrete måleverdier. Vi kan med skåring rangere besvarelser ut fra de målbare verdiene ved å gi dem en karakter. Dette gjør oss i stand til å gjøre de sammenligningene som man trenger for å finne nivået til de subjekter vi studerer. Man kan sette sammen flere ulike skåringer i et regelverk, også videre kalt regelsett. Karaktersettere kan også kalles ”Gradere”, men i denne oppgaven benyttes kun konseptet ”karaktersetter”.

2.3.3 Test og eksperimenter

For å finne mer ut om ferdighetsnivået til subjektene lar man disse besvare oppgaver definert gjennom en test. En test lar disse subjekter utføre oppgavene i en predeterminert rekkefølge. Er den computer-adaptiv, er vanskelighetsgraden også stigende. Resultatet av testen gir indikasjon på ferdighetsnivået til subjektet, men man trenger eksperimenter for å finne ut om rekkefølgen på spørsmålene påvirker resultatet. Definisjonen på et eksperiment, er at det er et sett av observasjoner for å bekrefte eller avkrefte hypoteser eller forskningsspørsmål (Wikipedia.org 2008). En test organiserer et sett av oppgaver og tilhørende stimuli som kan utføres i bestemte rekkefølger, og defineres som en prosedyre for å observere og beskrive oppførsler til subjektene[22]. I motsetning til tester kan rekkefølgen på oppgaver i eksperimenter være vilkårlige, slik at man kan se hvordan endringer i slike rekkefølger påvirker resultatet. I tillegg finnes en tredje form for tester som kalles computer-adaptive tester. Rekkefølgen på oppgavene i slike tester kan endres i forhold til tidligere prestasjoner for subjektet. Dette temaet vil ikke bli drøftet i dypere detaljer videre i denne oppgaven, men trekkes inn igjen i diskusjonsavsnittet.

For å sammenligne resultater av tester er det flere referansemetoder som benyttes, og jeg trekke frem kun et par av disse: **Normreferanse**[23] (Wikipedia.org 2008) er når man sammenligner resultatet med resultatene med en norm. En norm dannes ved at man lar en referansegruppe utføre testen før den gjøres tilgjengelig for flere[24]. En annen metode er **kriterium referanse**[25] (Wikipedia.org 2008): I

[22] Subjekt: En person som deltar i en test eller et eksperiment.

[23] Norm-referenced test." fra http://en.wikipedia.org/wiki/Norm-referenced_test

[24] Kilde: Bond, Linda A, ”Norm- and Criterion-Referenced Testing”, 1996, ERIC Identifikator: ED410316, fra http://www.ericdigests.org/1998-1/norm.htm, 6. juni 2008

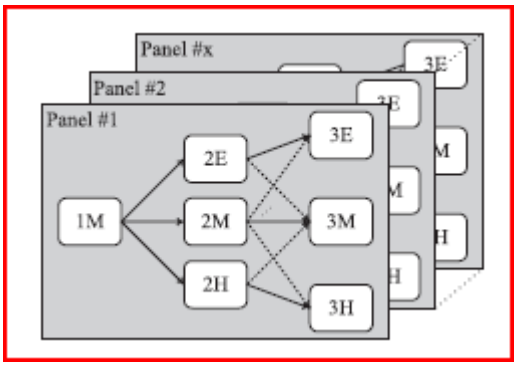
slike referanser ser man på om besvarelsen tilfredsstiller visse kriterier. Man opererer gjerne med ”cut score” i forbindelse med denne typen, det vil si et minimumskrav til hvor stor andel man skal ha riktig for å få oppgaven godkjent.

2.3.4 Item

Oppgave item er sammensatt av en beskrivelse om hva oppgaven går ut på, samt spørsmål og informasjon knyttet til denne. Man snakker om flere ulike oppgavetyper. Noen oppgaver ber subjekter om å skrive et essay, mens andre igjen kan be subjektene om å lage et program. Man kan også be subjekter om å rette opp feil i programkoder. I programmeringssammenheng er det de to sistnevnte oppgavetypene som er de mest interessante, fordi man kan hente en god del informasjon ut fra besvarelser av disse oppgavetypene. Subjektene kan enten løse bestemte problemer forbundet til programmering (analyse, design, implementering) eller til retting av feil (vurdering av feil, løsing av oppgave). Stimulansene subjektene får, defineres gjennom oppgavetekster og eventuell tilleggsinformasjon, og formidles gjennom SESE eller andre front end. Et item består av beskrivelse og et innhold som subjektene kan svare på. Oppgavene definerer vanligvis også eventuelle avgrensninger slik som tidsforbruk. Avhengig av oppgaven kan ulike respons gis av subjektene og skiller man også mellom ulike oppgavetyper. Fordi ulike metoder for evalueringer og skåringer knyttes sterkt opp til oppgavens innhold, må disse relasjonene tas vare på. Samlinger av oppgave item tas vare på gjennom en egen oppgavebank (item bank).

Deloppgaver, eller ”testlet”, er spesielle oppgaver hvor spørsmålene bygger på hverandre, og kan dermed gi statistisk avhengigheter i skåringene av delsvarene. Med statistisk avhengighet mener man at man at dersom en person løser en bestemt oppgaver økes sannsynligheten for at neste oppgave også løses. I forbindelse med adaptive tester, kan man få ulike vanskelighetsgrader på de ulike deloppgavene.

[25] Fra http://en.wikipedia.org/wiki/Criterionreferenced_test., Sist besøkt 28. april 2008.



Figur 1 Progresjon i testlet.
Kilde: (Luecht, Brumfield et al. 2006)

Slik man ser i fig. 1, kan man ha ulike progresjoner gjennom oppgaver som er innordnet i flere mindre deloppgaver, når man i tillegg tar høyde for at de ulike deloppgavene kan ha ulike vanskelighetsgrad. Disse ulike variantene skal det ikke tas høyde for i denne oppgaven.

2.3.5 Item bank

I Item bank er en samling av oppgave item med beskrivelse og en identifisering. Oppgavene kan hentes fra denne gjennom å bruke en algoritme for å velge ut item. På engelsk kalles den ”item selector”, men en norsk oversettelse kan være ”itemutvelger”. Fordi dette er et konstruert begrep, er det heller ikke funnet noen annen oversettelse. Denne banken er i utgangspunktet ukalibrert, hvilket betyr at item i denne ikke er kalibrert eller kun gitt en preindikasjon av vanskelighetsgrad. Utvelgeren er en komponent som plukker ut neste item i test og eksperimenter gjennom ulike algoritmer, og benyttes på kalibrerte item bank i computer-adaptive tester. Det finnes ulike algoritmene for utvelgelse av item avhengig av om klassisk testteori eller item respons teori danner basis for algoritmen. I klassisk testteori velges de ut fra vanskelighetsgrad og diskriminering, men i item responsteori baserer vil den endelige valget baserer seg i tillegg på informasjon som de bidrar med i den totale testen (Hambleton & Jones 1993).

2.3.6 Computer-adaptive tester

Computer-adaptive tester (CAT) baserer seg på fem komponenter: 1) Kalibrert item bank, 2) man har et inngangspunkt til testen, 3) det finnes en algoritme for valg av oppgaver, 4) man har prosedyrer for skåring av besvarelser, og 5) man har kriterier for terminering av testen. Den grunnleggende algoritmen er at man søker igjennom etter en tilgjengelig oppgave basert på et estimat for evnene til subjektet. Deretter presenteres oppgaven for subjektet, og en tabell over evnen til vedkommende oppdateres etter at besvarelsen er evaluert og skåret. Samtidig må evneestimatet oppdateres på bakgrunn av de nye dataene. Dette er en prosess som kan gjentas helt til man når

avslutningskriteriene for testen (Wikipedia.org 2008). Forskjellen mellom tester og computer-adaptive tester ligger hovedsakelig i begrepet ”adaptiv”. I tillegg er utvelgelsen av neste oppgave basert på maskinelle algoritmer. Det medfører at man kan endre utvalget av item basert på estimater av ferdigheter. Estimatet dannes på bakgrunn av tidligere besvarelser subjektet har levert inn. I denne sammenhengen må man bruke begrepet ”ferdighet” fordi man ser besvarelsene i sammenheng med hverandre, og ikke bare individuelt, slik det gjøres i denne oppgaven. Fordi computer-adaptive tester plukker item basert på algoritmer, og på bakgrunn av ferdighetsestimater, kan vanskelighetsgraden fra item til item som presenteres for subjektet korrigeres etter hvert som korrekte eller feilaktige besvarelser mottas. Forskjellen mellom regulære tester og computer-adaptive, ligger i at adaptive tester tilpasses et estimert ferdighetsnivå tilhørende det subjektet som testes. Gjennom definisjon av eksperimenter kan man teste hvilke resultater man får dersom man endrer rekkefølgen på oppgavene, og dermed kan se påvirkningene som kan oppstå som følge av at oppgaver kalibreres i ulik rekkefølge. Ferdighetsestimatet for subjektet kan korrigeres etter hvert som disse besvarelsene vurderes som korrekte eller feilaktige. For å bidra til kalibrering av oppgavebanken, kan skåringsrapportene som lages som følge av denne oppgave benyttes i den sammenheng. Kalibrering av denne banken kan baseres på statistiske data gitt av ulike karaktersettere (de Gruijter 1986), og er en av forutsetningene for å kunne utføre computer-adaptive tester. Den viktigste statistiske egenskapen for hvert item, er å se på hvordan besvarelser for denne oppgaven skåres. Dersom subjekter hovedsakelig får lav skår på oppgaver, kan det skyldes at disse oppgavene er for vanskelig, eller at kriteriene for oppgaven må endres.

2.3.7 Programmeringsferdigheter versus prestasjonsnivå

Prestasjonsnivå innen programmering er målinger utført gjennom ulike kriterier for enkeltbesvarelser. Når disse settes i sammenheng kan man snakke om målinger av ferdigheter. Prestasjoner og ferdigheter måles gjennom å la subjekter delta i tester og gi besvarelsene en karakter. Karakteren kan dannes på bakgrunn av ulike kriterier. Man har flere ulike teorier i forbindelse med hvordan disse nivåene skal måles, og disse nevnes i avsnittet om testteorier.

2.3.8 Prestasjon og effektivitet

Prestasjon er en konkretisering av en hvordan ferdigheter kan måles. Programmere tenker som oftest på ytelse i forbindelse med hvor effektiv programkoden er. I sammenheng med vurderinger av prestasjonsnivået er tidsforbruket til subjekt mer interessante, med mindre man tester implementeringer av algoritmer slik man av og til gjør i konkurranser. Med prestasjon menes

nødvendigvis ikke hvor mye kode en programmerer genererer i løpet av en tidsperiode, men like mye hvor korrekt den er, i henhold til funksjonelle krav og til regresjonskrav.

2.3.9 Regresjonstesting, funksjonalitet og tidsforbruk/effort (RFE)

Regresjonstesting – regresjon betyr å vende tilbake til det kjente, og hensikten med å benytte denne finne ut om endringer i program introduserer nye feil. Den anvendes etter at et program er modifisert, og trenger en referanse som den kan sammenlignes mot. Ved å ta vare på korrekte besvarelser og sammenligne disse med nye, vil man finne ut om nye feil introduseres. Til sammenligning går regresjonsanalyse ut på modellering og analyse av statistiske data (Wikipedia.org 2008).

Funksjonalitetstestingen innebærer å se om programmet tilfredsstiller de krav som stilles til funksjonalitet i programmet, med andre ord korrekthet. Funksjonaliteten testes normalt sett med å sammenligne output og metoderesultater med forventet resultat.

Tidsforbruk (”effort”) vektlegger hvorvidt tidsforbruket på oppgavene er i henholdt til begrensinger for bruk av tid. Dette benyttes for å se om hvorvidt subjektet har holdt seg til angitt tid, eller om den overskrides. Tiden kan deles inn i lesetid, samt tid til å utvikle løsninger for de ulike besvarelser som inngår.

2.4 Andre begreper

I denne oppgaven er det også en del andre begreper som benyttes. Her er de viktigste.

2.4.1 Brukere

I denne oppgaven defineres konseptet ”Bruker” som en person eller rolle som kan utføre operasjoner på ulike komponenter i et rammeverk. En ”rolle” er noe man tilordnes med den hensikt å utføre bestemte oppgaver. Disse kan ha tilgang til rammeverket, gjennom metoder for å autentisere[26] brukeren, og organiseres på en egen måte.

[26] Autentisere: Bekrefte at en bruker er ekte (autentisk).

2.4.2 Managers

Managers er et abstrakt begrep som betegner alle administrative deler i et system. Alle slike komponenter har ansvaret for underliggende komponenter i rammeverket, slik at disse kan nås fra høyere abstraksjonsnivå. Disse kan ha koblinger som knytter sammen ulike komponenter i et rammeverk.

2.5 Mer om computer-adaptiv testing

Bakgrunnen for computer-adaptiv test er problemstillinger i testteori for hvordan man kan måle ferdigheter hos personer. Disse målingene skjer gjennom å la subjektene starte med et utgangspunkt basert på estimater av nivået. Oppgavene som gis hentes ut via itemutvelger basert på ulike algoritmer. Disse algoritmene må hente oppgave item fra en kalibrert item bank. Banken kan blant annet kalibreres ut fra statistikker for hvert item som inngår i denne. Statistiske data hentes ut fra hvordan skår fordeler seg på de ulike oppgavene (Hambleton & Jones 1993).

2.5.1 Item utvelgelse og item bank kalibreringer

Utvelgelse av item baserer seg på gjette hvilken oppgave som bør velges neste gang basert på hvordan subjektet har svart tidligere. For å estimere dette må man først estimere vanskelighetsgraden på oppgaven. Dette gjøres gjennom en prosess kalt item bank kalibrering. Alle oppgaver i bankene starter enten som ukalibrerte eller ukalibrert, men gitt en preindikering på vanskelighetsgrad. Videre gis de en parameter for hvordan disse skilles. Disse, samt en parameter for å gjette vanskelighetsgraden, benyttes for å beregne den reelle vanskelighetsgraden ut fra skårstatistikker for de enkelte item. I formlene som inngår i dette, gjøres det enkelte antagelser om fordeling av de statistiske dataene (de Gruijter 1986). Dette kan gjentas oppgaven er ferdig og oppgaven har gått inn i en pensjonert tilstand. Gjennom å estimere ferdighetsnivået til subjektet, og å sammenligne det med vanskelighetsgraden på ulike item, kan man finne det item som tilnærmelsesvis passer best som neste oppgave for subjektet. Alle oppgavene startes enten som ukalibrerte, eller ukalibrert hvor det er gitt en preindikering på vanskelighetsgrad. Videre gis de en parameter for hvordan disse skilles. Disse, samt en parameter for å gjette vanskelighetsgraden, benyttes for å beregne den reelle vanskelighetsgraden ut fra statistikker.

2.6 Begrensninger i eksisterende rammverk

I forberedelsene til denne oppgaven kikket jeg nærmere på flere av de rammeverk som eksisterer og som benyttes, både hos universiteter og i kommersiell sammenheng. Det jeg da la merke til var at flere av disse sterkt begrenses av de formål de er designet for, blant annet med henblikk på programmeringsspråk, og metoder for evaluering og skåring av besvarelser. De skiller heller ikke mellom konseptene evaluering og skåring, noe som trengs i forbindelse med automatisk vurdering av besvarelser. Fordi man vet lite om metodikkene som ligger til grunn, vet man heller ikke så mye om hvordan de kommer frem til de skåringer de gir. I testteorier finnes noe som kalles computer-adaptive test. En av de største begrensningene ved eksisterende rammeverk, er at flere av dem mangler støtte for computer-adaptive tester. Mange er også lite generelle i forhold til de oppgaver man tester eller de ikke er plattformuavhengige. Ser man i tillegg på de kommersielle rammeverkene som eksisterer, er de ikke bare kostbare, men baserer seg ofte på proprietær programkode noe som gjør det vanskelig å kunne benyttes i forskningssammenheng.

3. Utviklingsprosesser og metoder.

3.1 Testmetoder

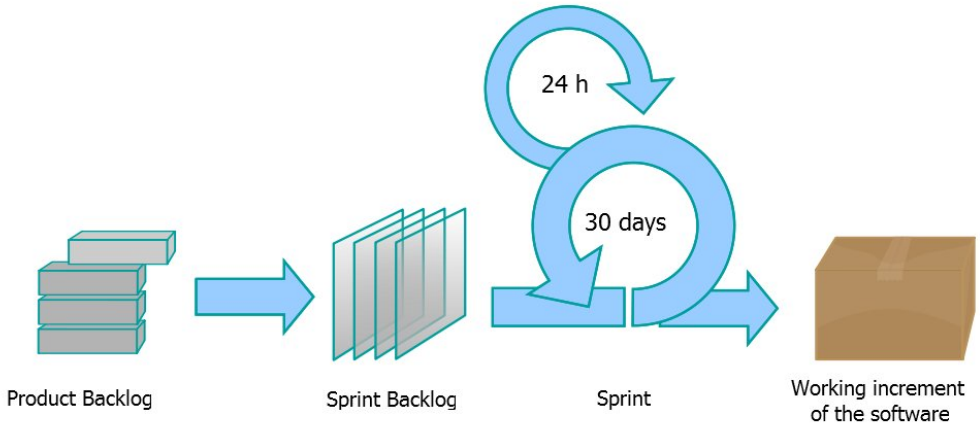
I denne oppgaven er det benyttet ulike metoder for å teste de resultater som man er kommet frem til. Resultater av metoder som er lagt til er testet opp mot referanseresultater. Utvidbarhet i rammeverket er testet gjennom å ha lagt til nye evalueringsmetoder av programmeringsbesvarelser. Disse er benyttet for å se om implementasjon og metodikk er korrekte.

3.2 Testdrevet utvikling

Bruk av testdrevet utvikling har sitt utspring i en programmeringsretning som kalles ”Extreme Programming” (XP) som baserer seg på ”agile” utvikling av programvare i små grupper, men som etter hvert har blitt inkorporert i profesjonell sammenheng og i utdannelse (Edwards 2003). Innen testdrevet utvikling er bruk av enhetstesting for å kontrollere at implementeringen tilfredsstiller bestemte krav forholdsvis vanlig.

3.3 SCRUM

SCRUM er en utviklingsmetode for bruk i agile utvikling av programvare (Jim Highsmith & Cockburn 2001) for små grupper hvor personer i gruppene har ulike roller. Gruppen ledes av en SCRUM mester/ prosjektleder som styrer et utviklingsteam. Noen av fordelene med SCRUM er bruk av hyppige møter. Dette gjør at man kan utveksle informasjon om de oppgaver man holder på med og få innspill fra andre på mulige alternative løsninger på problemstillinger. I dette prosjektet har vi holdt oss til maks ett eller to møter i uken. Flere bedrifter ser ut til å bruke SCRUM som utviklingsprosess, fordi man kan utveksle informasjon om hvilken arbeidsoppgave man holder på med slik at man i mest mulig grad unngår at flere utviklere holder på med samme oppgave uten å være klar over det. Samtidig gir det muligheter for å diskutere mulige løsninger. Vanlige SCRUM team er på 5-6 personer, men det finnes en variant som kalles ”solo scrum” med færre personer involvert. Som man ser av fig. 2 foretas det sprint på mellom 20 timer og 30 dager. For hver sprint lages det en backlog (arbeid som venter på å bli gjort) som igjen er laget på bakgrunn av en backlog for produktet.



Figur 2 SCRUM utviklingsprosess.
Kilde: http://en.wikipedia.org/wiki/Scrum_(development)

3.4 SMART dokumenter

SMART dokumenter er dokumenter som inneholder informasjon om de oppgavene man ønsker å få gjort i henhold til *Specific*, *Measurable*, *Attainable*. *Realistic*, og *Time-bound* (Wikipedia.org 2008). På norsk oversettes dette til *Spesifikke*, *Målbare*, *Oppnåelig*, *Realistiske*, og *Tidsavgrenset* mål. Dette er en teknikk som har sin utgangspunkt i ”eXtreme Programming” (XP), men som brukes også i større prosjekter. Med ”spesifikk” menes at de målene man setter seg skal i mest mulig grad spesifiseres, men samtidig bør det være rom for litt fleksibilitet. Arbeidene man utfører skal kunne måles, for eksempel i form av at det er målbare forbedringer. De må også kunne oppnås; Man bør ikke sette seg mål som ikke lar seg uføre i løpet av den tidsrammen man har til rådighet.

3.5 Verktøy

Utvikling: Til utviklingen av programmet, er Eclipse med Java versjon 1.4.2 til 6.0 valgt. Da prosjektet startet opp ble Java 1.4.2 benyttet, men etter at det kom nye kompilatorer og at vi fikk sjekket at FIT fungerte med nyere versjoner av Java, bruker vi per dags dato Java 6.0.

Modellering: Til modelleringsarbeid i forbindelse med UML er Rational Software Modeller valgt.

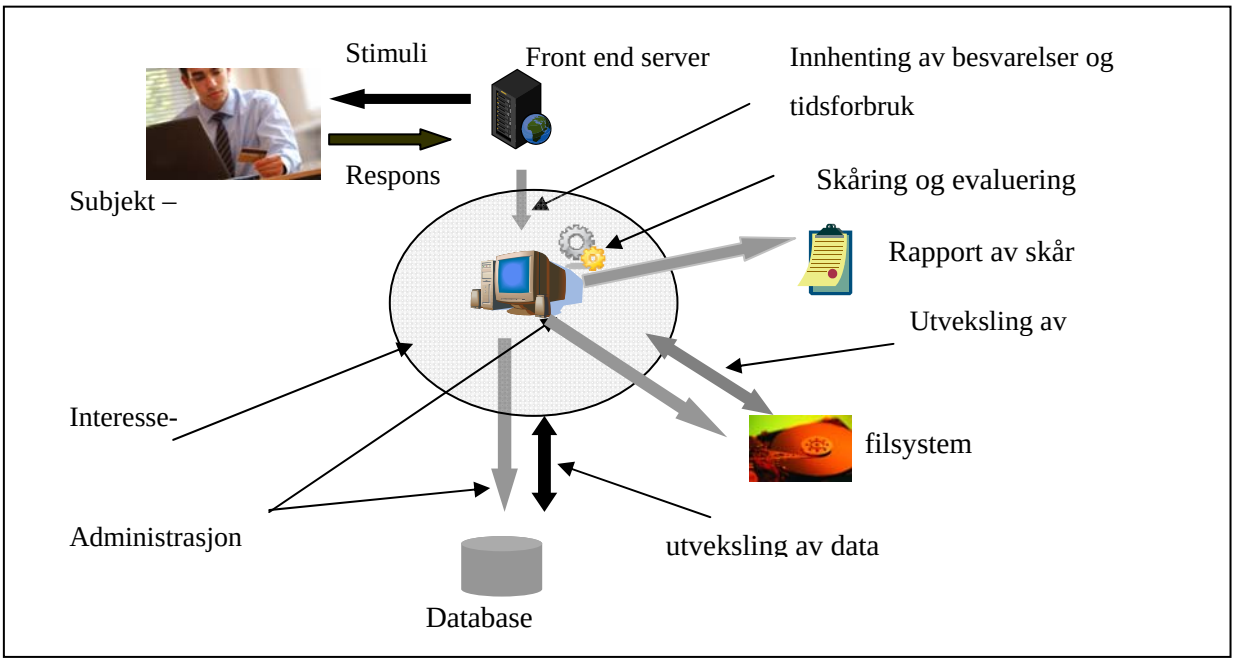
Versjonshåndtering: Versjonsverktøyet Tortoise SVN (tigris.org 2008) blitt benyttet for å ta vare på ulike versjoner av kildekoden.

4. Analyse av problemdomenet og kravspesifikasjon

Dette avsnittet tar for seg analyse av problemstillingene administrasjon, evaluering, delsvar, og rapportering, og å komme med en spesifikasjon av krav som må stilles til design og implementeringer. Kravene kan deles opp i funksjonelle og ikke funksjonelle.

4.1 Administreringsprosesser i et rammeverk:

Rammeverket i fig. 3 er plassert i midten fordi dette representerer sentrale deler av oppgaven. Dette rammeverket mottar besvarelser fra subjekter via SESE. Denne front end serveren kan, i tillegg til å motta besvarelser, også sende ut oppgaver med spørsmål til subjektene. Besvarelsene som kommer tilbake, må administreres for videre evalueringer og skåringer som skal utføres. Etter hvert som kriterier og metoder endrer seg, eller man legger til nye menneskelige karaktersettere, trenger man sammenligningsmuligheter for å gi informasjon om potensielle feil eller avvik. Disse kan presenteres gjennom ulike rapporter på ulike format.



Figur 3 Rammeverkets plassering i forhold til interesseområde.

4.2 JCAT: Utgangspunktet for oppgaven

Utfordringene for dette rammeverket, er knyttet til hvordan man skulle administrere og hente inn større mengder besvarelser som mottas via front end. Tidligere løsning baserte seg på ren manuell administrering, noe som er tidkrevende. Man hadde heller ingen måter å knytte sammen informasjon om tidsforbruk sammen med besvarelsene, slik at man kunne ta denne informasjonen med i vurderingene av besvarelsene. Det fantes heller ingen gode, effektive måter å organisere og vurdere delsvar på. Man trenger også å ha muligheten til å utvide evalueringsmetodene i forhold til de som allerede eksisterende. Eksisterende metoder bør studeres for å finne eventuelle fellestrekk, og for å se hvordan man i så fall kan prøve å gjøre nytte av disse. I utgangspunktet, når besvarelser med feil ble evaluert, ble det avdekket at skåringsresultatene for de etterfølgende besvarelsene også ble påvirket.

4.3 Generell beskrivelse av funksjonelle krav

Det stilles følgende funksjonelle krav til rammeverket:

Besvarelser: Alle besvarelser for hele eller deloppgaver skal administreres. De skal på bakgrunn av dette kunne evalueres og skåres effektivt. Rammeverket kan ikke avgrenses til et bestemt antall besvarelser.

Rammeverket må støtte nye og eksisterende metoder for evalueringer og skåringer. Automatiske evalueringer og skåringer må skilles fra hverandre, slik at man lett kan rekalkulere skår på bakgrunn av nye evalueringsmetoder og -kriterier.

Skår fra ulike karaktersettere skal presenteres i et tabellformat, men kravet reduseres til at man i første omgang lager en implementering som kun gjelder for en enkelt karaktersetter. Rammeverket må også ta høyde for semi-automatisk og manuelle evalueringer og skåringer**.** Da man trenger også semi-automatisk skåring, må manuelt satte karakterer også være tilgjengelige for rammeverket.

Tidsforbruk må kunne inngå i vurderinger som gjøres av besvarelser, for å kunne rangere besvarelsene bedre. Dette er viktig for å skille mellom subjekter som løser besvarelsene på angitt tid, og de som ikke gjør det.

Data hentet fra et pilotprosjekt utført i perioden november 2007 – mars 2008, hvor enkelte testdata er hentet fra, skal kunne administreres og vurderes gjennom de metoder som denne oppgaven skal implementere. Skåringer som beregnes på bakgrunn av disse skal kunne settes opp på tabellformat for sammenligning av ulike karaktersettere.

4.4 Administrasjon

Administrasjonen av besvarelsene må deles inn i to underkategorier. Den første kategorien er innhenting av besvarelser. Med dette menes innhenting av besvarelser fra SESE til JCAT. Det andre er organisering av besvarelser og tidsforbruk. Det tredje, er at man trenger å ha administrative rutiner for å sammenligne skår.

4.4.1 Innhenting av besvarelser

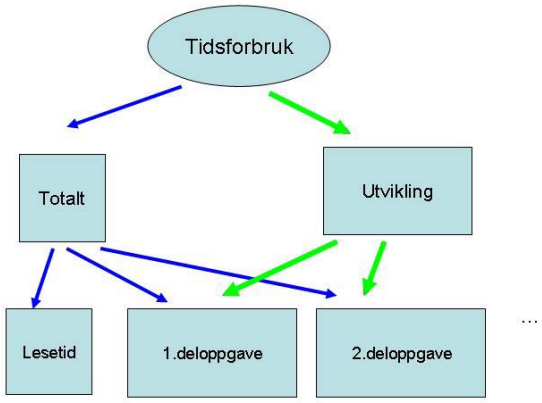
Innhenting av data fra SESE må skje i to faser. Den første fasen tar utgangspunkt i innhenting av besvarelser fra front end for deretter å legge dem over på et område hvor man lett kan hente dem inn. Sammen med disse besvarelsene hentes også inn andre data slik som tidsforbruk, også plassert på et veldefinert område, hvor de så legges ut på et tilordnet område. Fase to skal hente besvarelsene og tidsforbrukene inn i JCAT, slik at man kan utføre evalueringer og skåringer av disse. Samtidig tillates det at man kan kontrollerer besvarelsene og tidsforbrukene etter fase en. Man skal da kunne velge hvilken del man ønsker å få importert, men fordi tidsforbruk ikke kan eksistere uten besvarelser, velges å definere følgende sekvens: Først importeres besvarelser, og deretter importeres tidsforbruk. To identiske svar skal ikke forekomme, og dersom det finnes en besvarelse fra før av som er identisk med den nye, vil den eksisterende besvarelsen bli beholdt. Besvarelsene må kunne gjelde hele oppgaver, eller være delsvar som inngår i en større oppgave. Fordi navngivingen i front end og rammeverk ikke er identiske, må enkelte egenskaper døpes om slik at man kan forholde seg til begge. Besvarelsesbanken har ansvaret for å lagre de besvarelser som hentes fra front end via dataimporteringen. Den må håndtere at et subjekt kan levere inn flere besvarelser for et item og hente inn siste besvarelse først, men det kreves ikke versjonshåndtering av besvarelser da dette blir for omfattende for denne oppgaven. Det må også kunne utføres oppdateringer og slettinger, med tanke på å renske opp persistent lager. Bortsett fra å kontrollere om to besvarelser er identiske, blir ytterligere konflikthåndteringer for omfattende til å kunne tas med i denne oppgaven.

4.4.2 Organisering av besvarelser

Alle besvarelser kan administreres på ulike nivåer. I et rammeverk defineres besvarelsesbanken som en manager for besvarelser. Dersom man kun identifiserer besvarelser gjennom item navn og et subjekt navn, vil dette skape problemer når delsvar skal administreres da disse har samme item og subjekt identifikatorer, og strengere identifiseringsegenskaper kan være ønskelig. Heller ikke et filnavn, dersom besvarelsen er en fil, er unik nok til at den kan benyttes til unik identifisering fordi slik navngiving ikke garanterer at filnavnet er korrekt. Dersom man benytter en kode som identifiserer deloppgave i kombinasjon med overnevnte egenskaper, vil dette være sterkere identifikasjon. Ved å legge den ser man at man, vil man få en tilstrekkelig unik identifisering av de ulike besvarelsene, uansett om de gjelder hele eller delvise oppgaver.

4.4.3 Tidsforbruk

Tidsforbruket måles vanligvis i antall hele minutter. Man er ikke interessert i informasjon om sekunder i denne sammenhengen, men man har en utfordring dersom et subjekt leverer inn alle sine besvarelser på slutten fremfor å levere dem inn etter hvert som en underoppgave er unnagjort. Data hentes inn fra front end og må knyttes til de bestemte besvarelser definert av item, subjekt og besvarelser. Ikke bare innbærer dette at man kun måler totalt tidsforbruk, men man måler også tidsforbruk å besvare de ulike deloppgaver. Dersom tidsforbruket ikke registreres korrekt, får man utfordringer i hvordan slike situasjoner skal håndteres.

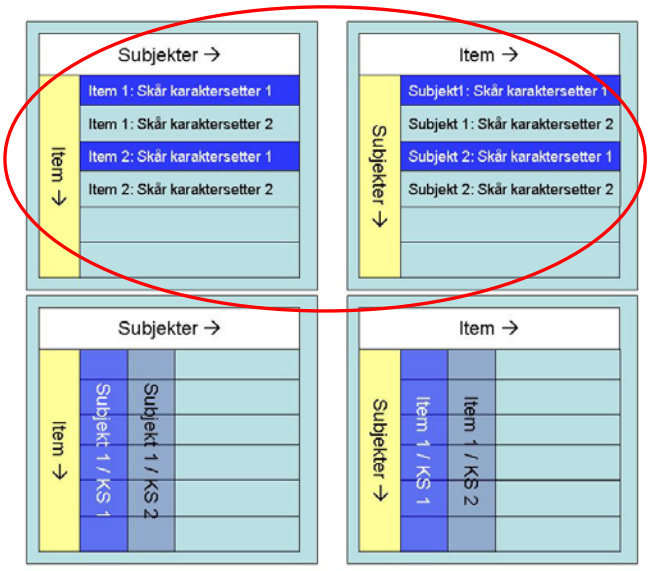


Figur 4 Ulike måter å vurdere tidsforbruk

Figur 4 viser to måter man kan tolke tidsforbruk på. Den ene måten, representert med blå piler, forholder seg til all tid som går med til besvarelsen av oppgave. Den andre måten, representert ved grønne piler, forholder seg kun til den tid som går med til den konkrete implementeringen av besvarelsene.

4.4.4 Sammenligning av skåringer

Generering av skåringsrapporter og kjøring av differanse verktøy velges å knyttes opp til brukere av rammeverket. For å kunne sammenligne skår laget av ulike karaktersettere, må man vite hvilken karaktersetter som gav skåren. Disse skår må tas vare på, og det finnes flere ulike alternativer for å gjøre dette. Man vil også kunne beregne ny skår i den forbindelse for å sammenligne gamle kriterier med nye. Det er to måter å gjøre slike sammenligninger på: Den ene har subjektene plassert horisontalt med alle item i vertikal retning, hvor man har en linje per karaktersetter per item. Den andre løsningen er å plassere item i horisontal retning og subjekter i vertikal retning, hvor man har en linje per karaktersetter per subjekt. Man kan også tenke seg at et item fordeles på to eller flere kolonner, dersom det finnes flere karaktersettere. Utfordringene med slike tabeller er å få mest mulig informasjon uten at det går ut over lesbarheten, spesielt når man får store mengder data som skal legges inn i disse. Man må gjøre et valg mellom disse to, og i denne oppgaven blir det kun fokusert på enkle tabeller med maks en karaktersetter.



Figur 5 Ulike formater på skåringsrapporter. Krav til implementering angis med rød ring.

For å kunne gjøre flest mulige sammenligninger, er det minst fire måter å lage disse skåringsrapportene på. Karaktersettere kan plasseres etter hverandre enten radvis eller kolonnevis, slik det illustreres i figur 5. Kravet er at implementeringen som skal lages, kan gjøre det mulig med små forandringer å tillatte alle disse formatene. Spørsmålet er nå hvem eller hva som skal ha ansvaret for at slike rapporter kan genereres. Dersom tabellene blir for store til at de er lesbare, må de deles opp. Cellene i tabellen kan inneholde informasjon av ulike typer. Ved rapportgenerering av skår for sammenligning må skårene beregnes på nytt. Man kan velge å først iterere over alle oppgaver, og for hver oppgave item kan det itereres over hvert subjekt eller motsatt. Hver besvarelse må hentes frem for nye beregninger. Dersom man skal skåre på bakgrunn av nye kriterier, må man ta vare på de gamle verdiene, slik at de opprinnelige innstillingene kan gjenopprettes. Fordi man skal tar vare på skår laget av ulike karaktersettere, trenger man å knytte skåren opp til de enkelte besvarelser for de enkelte subjektene. Man trenger informasjon om alle subjekter når rapportene skal genereres, og det er dermed nyttig å lagre dem i subjektbanken[27], hvor de kan tas vare på og hentes ved behov.

4.5 Evaluering og skåring av besvarelser

Vurderinger av besvarelser bør ikke påvirkes av at besvarelser kan ha ulik struktur. Dersom man ser på disse besvarelser som bokser hvor man kan hente ut informasjon eller stimulere, uten å ta hensyn til innhold, kan man få mer korrekte resultater. Det konseptuelle termen for dette er ”svartbokstesting”[28]. Likevel kan det i noen tilfeller være ønskelig å teste ut enkeltkomponenter i slike bokser, for å se om de oppfyller bestemte krav. I slike situasjoner kan det være nødvendig å åpne opp en boks for å få tilgang til slike komponenter. Ved å teste disse komponentene, kan man få mer informasjon om hvorfor en bestemt karakter ble gitt. I tillegg til å kunne legge til nye kriterier, må man noen ganger også endre vektleggingen av de ulike kriteriene, i tilfelle det viser seg at enkelte vektlegges for mye eller for lite.

Alle skåringsmetoder som skal legges til, må kunne gi en skår tilbake basert på evalueringer av de besvarelser som gis. Hvordan de kommer frem til resultatet kan variere, men et fellestrekk er at de baserer seg på delskår beregnet på basis av evalueringsmetodene som ligger i bunnen. Dette gjelder like mye metoder basert på menneskelige ressurser, som de som baserer seg på bruk av automatikk. I utgangspunktet ser jeg på hvordan man kan legge til menneskelig skåring, regelsett og andre evalueringer og skåringer av besvarelser. Når en programmeringsbesvarelse skal evalueres, må den

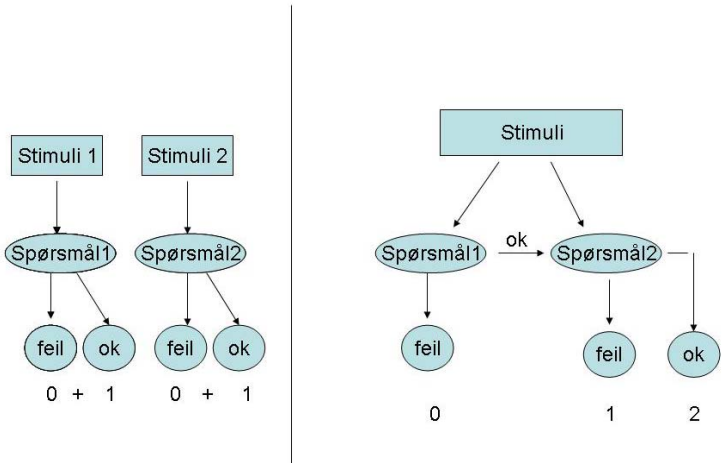
[27] Subjektbank: en samling av subjekt.

[28] Svartbokstesting: En metode for å generere testcase som er uavhengig av programvarens struktur. Kilde: Encyclopedia of Software Engineering Vol. 1, 2002.

først pakkes ut. Etter at dette er utført må den kompileres. Her kan man risikere at det oppstår feil som må håndteres og informeres om videre, da dette har betydning for resultatet. Bakgrunnen for at en evaluering kan utføres er at det er hentet inn en besvarelse som initieres som igjen medfører at nødvendige omdøpinger gjøres, og deretter kompileres. Deretter utføres en testcase som benytter seg av ulike metoder på den koden som skal evalueres. Denne testcasen kan basere seg på eksisterende evalueringer, slik som FIT eller JUnit, eller på fremtidige evalueringer slik som softwaremetrikk[29]. Alle slike metoder kan legges til gjennom ulike teknikker, men man må av og til tilpasse disse til de ulike metodene som implementeres. Et eksempel på en slik tilpasning er bruk av enhetstester som evalueringsmetode. Fordi resultatet ikke kommer som en vektor, men som en formattert tekst må de ulike verdiene plukkes ut på spesielle måter.

4.5.1 Delsvar

Som man ser tidligere finnes det flere ulike vurdere oppgaver på. Når man skal vurdere delsvar, må det tas hensynt til om spørsmålene er uavhengige, eller om de baserer seg på tidligere spørsmål. Dersom det siste er tilfellet, vil resultatet av vurderingen av forrige delsvar avgjøre om neste skal vurderes, slik figur 6 viser i høyre halvdel. Dette skal også gjelde for etterfølgende oppgavespørsmål.



Figur 6 Skåring av delsvar.

Skåring av delsvar skal resultere i ulike karakterer, avhengig av om spørsmål 2 er basert på spørsmål 1 eller ikke, slik det vises i høyre halvdel av figur 6. I venstre halvdel er spørsmål 2 uavhengig av spørsmål 1 mens i

[29] Softwaremetrikk: målinger som kan hentes fra egenskaper ved livssyklusen til et program. Kilde: Encyclopedia of Software Engineering, Second Edition, Vol 2, Wiley & Sons Inc, New York, 2002. ISBN: 0-471-21007-2.

høyre halvdel må man svare korrekt på spørsmål 1 for å få en karakter større enn 0. Alle besvarelser må administreres slik at hvert enkelt delsvar kan identifiseres. Selv om filnavnet til besvarelsen ikke i utgangspunktet ikke nødvendigvis unik, kan en omdøping av dette likevel gjøre filnavn til en unik identifikator, forutsatt at det gir tilstrekkelig informasjon. Tilstrekkelig informasjon kan være å angi hvilken deloppgave besvarelsen gjelder. Når man skal vurdere slike besvarelser, må man vurdere hvorvidt man skal ta utgangspunkt i skåring eller evaluering. Dersom man tar utgangspunkt i skåringen, kan man lage en samling av ulike delskår av svar for de enkelte deloppgavene med tilhørende evalueringer, hvor hver evaluering kan være av ulik karakter eller av samme type. Ved å definere et overordnet regelsett for hvordan disse skal relateres, kan man gi ulik skåring avhengig av hvor mange deloppgaver som er blitt besvart og alternativt om man har feilet på en av underoppgavene. Hvis man begynner med utgangspunkt i evaluering, må en evaluering gjelde for en eller flere besvarelser av deloppgaver, men man mister dermed litt av de egenskapene som man ellers ville ha fått. På bakgrunn av dette velges skåring som utgangspunkt.

4.5.2 Regelsett

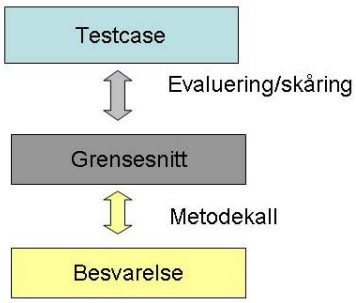
Som jeg kommer til å snakke mer om, kan et Item deles inn i underoppgaver hvor man kanskje ønsker ulik skåring eller evaluering på de ulike delene. Ved å definere et sett av evalueringer, delskåringsmetoder og skåringsmetoder, og gjennom å benytte et regelsett kan man gi regler for hvordan den totale skåren til slutt kan beregnes. I tillegg til å kun vurdere rene funksjonelle krav, kan den også ta hensyn til tidsforbruk og andre krav som stilles til oppgavebesvarelsen. Dette regelsettet bør i teorien være i stand til å skille mellom tid forbrukt på å lese oppgaven, og tid som benyttes til å løse den. Regelsettene må lett kunne endres, og være fleksible med tanke på nye måter å vurdere besvarelser.

4.5.3 Håndtering av menneskelig skåring

En menneskelig karaktersetter i denne forbindelsen, som jeg nevnte tidligere, er en person som kan sette en skår på hele eller deler av oppgavebesvarelser. For å kunne sammenlikne disse med resultater av de automatiske skårene må disse tas inn. Karaktersetteren kan også sammenligne besvarelser mot hverandre og bruke denne informasjonen i sine vurderinger. Vurderingene som gjøres kan også benytte delvis automatiske metoder for å effektivisere enkelte deler.

4.5.4 Uavhengighet

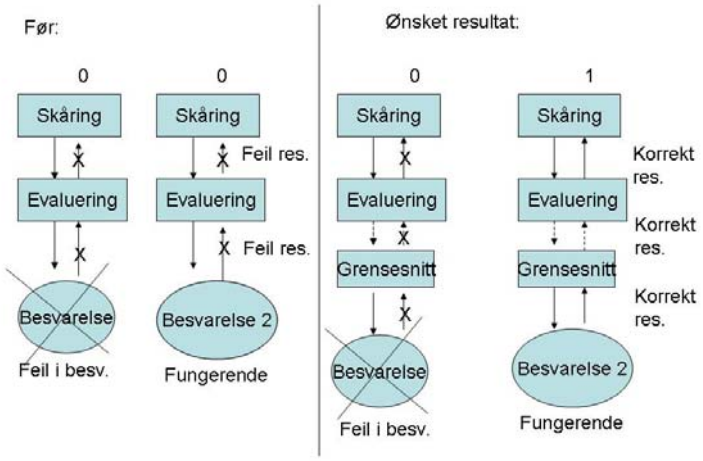
Det trengs å tilbys en evalueringsmetode som danner et grensesnitt mellom testcase og den testede besvarelsen som kan endres i kjøretid og dermed ikke låses ved kompilering. For å kunne fungere må den laste inn klasser etter behov, og kalle de metoder som skal vurderes. I et tilsvarende rammeverk FIT[30] finnes lignende eksempler som benyttes allerede, men disse kan ikke benyttes direkte fordi de enten ikke er tilgjengelige. En annen grunn er de er spesifikt tilpasset andre formål. Utfordringen man står overfor her å kunne identifisere klasser og metoder som man ønsker å teste. Dette er tilsvarende det problemet man får dersom man ønsker å utføre vurderinger av besvarelser som ufullstendige eller på formater man ikke har tatt høyde for i testcase.



Figur 7 Plassering av et grensesnitt i forhold til testcase og besvarelse.

Ved å plasseres dette grensesnittet mellom testcase og besvarelser, slik det vises i figur 7, kan evaluering- og skåringsmetoder i stedet forholder seg til den. I motsetning til å forholde seg direkte til besvarelser, vil testcasene ikke ha kjennskap til feil som oppstår før den utfører metodene som skal testes. Gjennom innkapsling av unntak, vil unntakssituasjoner som oppstår håndteres i kjøretid fremfor under kompilering av rammeverket.

[30] FIT: http://www.fitnesse.org/



Figur 8 Feil i skår forårsaket av tidligere besvarelser

Som man ser av figur 8, er hensikten å hindre at feil i en besvarelse (representert ved besvarelse 1) påvirker skåringen av andre besvarelser (representert ved besvarelse 2), slik det vises i venstre halvdel. De feil som oppstår på bakgrunn av besvarelse 1 ønsker man å oppføre seg, slik det vises i høyre halvdel. Besvarelse 2 skåres korrekt selv om besvarelse 1 inneholder feil som gjør at den ikke kan evalueres og skåres automatisk. Det er vanskelig å bevise at det vil holde i absolutt alle tilfeller, men man øker sannsynligheten for at det vil også fungere i tilfeller man ikke har sett på. Fordi den skal skille bedre mellom testcase og besvarelser, må unntakssituasjoner også håndteres. Man har to strategier for dette: Det ene er innkapsling av unntak, og det andre er å gjøre ingenting og håpe på det beste. Dersom man ikke sender unntak videre, men i stedet fanger dem opp, kan man risikere feil i skår fordi skåringer også kan basere seg på antall unntak.

4.5.5 Utvidelse av vurderingsmetoder

Til vanlig benyttes JUnit, en bestemt type enhetstesting til å avdekke problemer i implementasjoner. En vanlig måte er å definere grensesnitt, slik at man kan benytte arv for å teste de individuelle implementeringene. En enhetstest defineres i testcase hvor ulike enheter kan testes. Når en enhetstest utføres er det fire egenskaper man ønsker å studere: Den første egenskapen er antall tester som ble utført, den andre er antall av disse som ble utført korrekt. Antall feil og antall tester som ikke kunne utføres, er også viktige egenskaper ved disse. Når enhetstester skal utføres, trenger man ta hensyn til om det skal utføres gjennom grafisk grensesnitt eller kommandolinjemiljø. Kravet for å benytte enhetstesting som evalueringsmetode, er at den bør kunne utføres gjennom kommandolinjemiljø, og at verdiene denne metoden produserer kan ekstraheres for videre skåring. Det er to formater som må støttes i denne sammenhengen: Det første formatet er dersom alle tester fungerer feilfritt, og det andre er dersom en eller flere tester medfører ulike feilsituasjoner.

4.6 Ikke-funksjonelle krav

I denne oppgaven stilles det følgende ikke-funksjonelle krav:

Åpen kildekode: Alle implementeringer skal gjøres i åpen kildekode.
Dokumentasjon: Det må skrives god dokumentasjon i kildekoden. Fordi man jobber i et multinasjonalt miljø ved Simula, må dokumentasjon av kildekode skrives på engelsk. Det er ikke satt krav om at denne oppgaven ellers skal skrives på dette språket.
Vedlikeholdbarhet og utvidbarhet: Komponenter må kunne testes og være enkle å vedlikeholde, og ny funksjonalitet må kunne legges til i systemet.
Plattformuavhengighet: Implementeringer må være minst mulig plattformavhengige, men dette er ikke et absolutt krav.

5. Design og implementasjon

I dette kapittelet ser jeg nærmere på designproblematikken, og beskriver hvilke alternativer som finnes i designet og hvilke valg som foretas. De første og underliggende problemstilling som må løses knytter seg til enkeltoppgaver, Disse løsningene blir benyttet når jeg diskuterer oppgavestruktur (hel/deloppgaver) samt regelsett for skåring og karakterer. Løsningene kan benyttes på besvarelsene som mottas via front end, og i den forbindelse ser jeg på metoder for administrering av ulike besvarelser for både hele og deloppgaver samt rapportering av skår for sammenligninger. I kapitel 5.1 ser jeg på hvordan man kan administrere ulike typer besvarelser. Kap. 5.2 omhandler metoder for hvordan disse kan vurderes, og i 5.3 ser jeg på strukturmessige endringer som er skjedd i rammeverket JCAT.

5.1 Administrasjon

5.1.1 Innhenting og organisering

Dataimportering er nødvendig for å tillate en manuell inspeksjon av koden før en automatisk evaluering og skåring blir utført. Dette gir også muligheter til å ta avgjørelser for å se om tidsforbruket virker å være fordelt riktig i henhold til oppgavens krav. For å vite hvordan man skal designe løsningen for administrasjon av besvarelser, må man vite hva identifiserer en besvarelse unikt, og hvordan flere besvarelser fra samme subjekt på samme oppgave håndteres i forhold til evaluering og skåring.

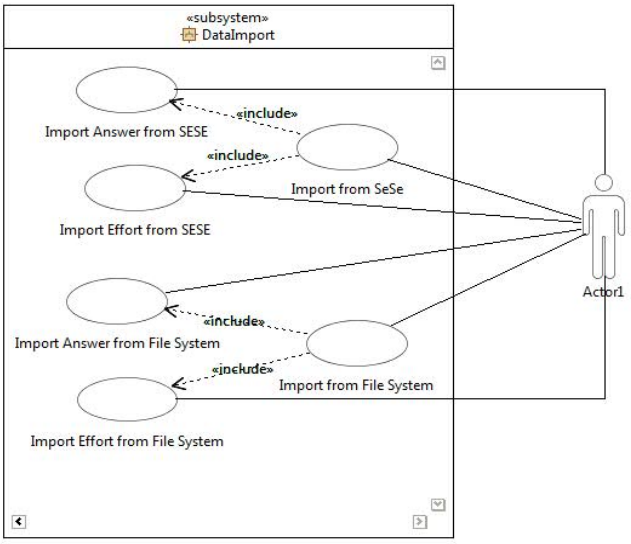


Fig. 9 Bruksmønstre for dataimportering.

Fig. 9 viser to hovedfaser for importering av besvarelser og tidsforbruk. Hver fase har separate deler for importering av besvarelser og tidsforbruk.

Fase 1, første del, skal importere innpakkede besvarelser fra front end og legge dem ut på en katalog tilordnet dette formålet i henhold til item og subjekt. Filer må omdøpes i henhold til de deloppgaver de gjelder. Andre del skal importere data om tidsforbruk fra front end, lagret i eget format, og overfører dem til filer plasseres og navngis i henhold til besvarelsen disse dataene gjelder.

Fase 2 overfører de data som ligger på fra katalogstrukturen over til en database, hvor besvarelsene senere kan hentes for evaluering og skåring. Etter at selve besvarelsen er hentet inn, kan også tidsforbruk legges til. Hver besvarelse som får endret data om tidsforbruk må også oppdateres i persistent lager, slik at disse kan tas med når evalueringer og skåringer skal utføres. Første del i denne fasen sørger for at databasen oppdateres med nye besvarelser. Deretter kan del 2 sørge for at disse oppdateres med informasjon om tidsforbruk innhentet i fase 1.

Informasjon om item, subjekt, deloppgaver og besvarelser ligger plassert i filen på følgende måte: Katalogene på øverste nivå inneholder item, deretter ligger kataloger som er navngitt i henhold til koder som identifiserer enkeltspørsmål i oppgaven. Under disse ligger igjen subjektene som egne kataloger, og bladnodene i treet er selve besvarelsene. Dette kan illustreres med utgangspunkt i roten på treet slik at man får en struktur som ser ut som i fig. 10. I denne figuren kan man legge merke til delsvar (1) og (2) som har samme navn selv om de gjelder to ulike deloppgaver.

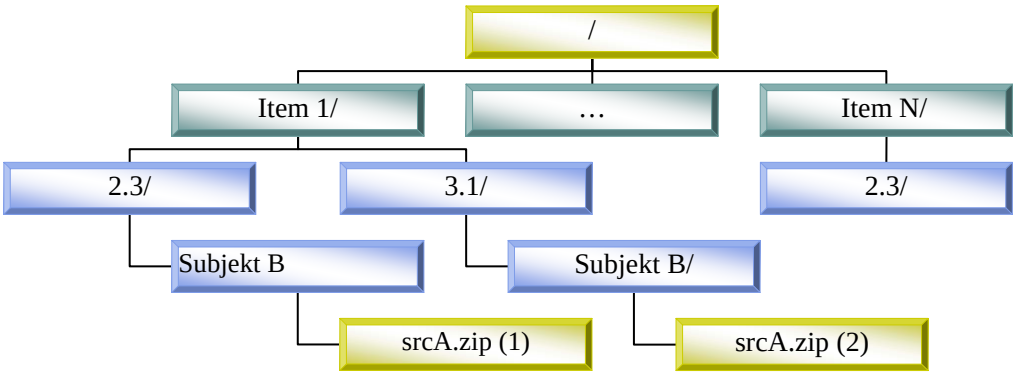


Fig. 10 Struktur mottatt fra front end.

Ved å omdøpe besvarelser for 3.1 til deloppgave B, kan man endre strukturen slik at hvert subjekt får alle sine respektive svar knyttet til seg, slik det vises i fig. 11.

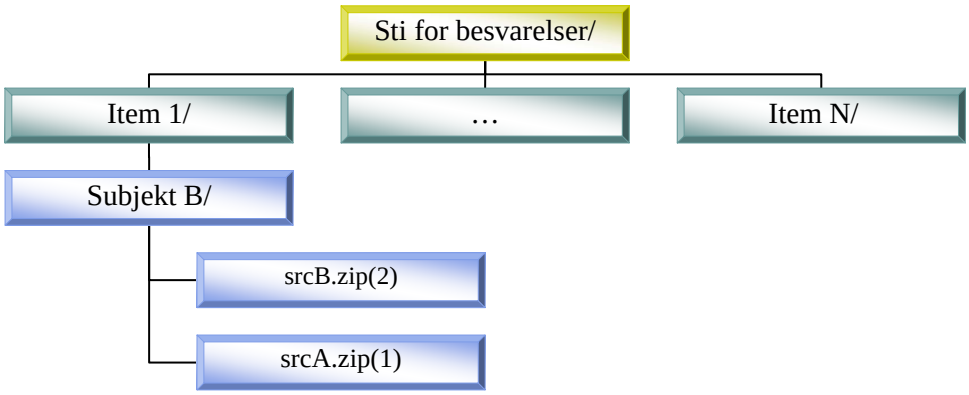


Fig. 11 Ny struktur etter importering til katalogstruktur.

I fig. 11 er delsvar (2) blitt omdøpt til ”`srcB.zip`”. For å unngå konflikter og potensiell inkonsistens i databasen må hver besvarelse, enten den gjelder en hel eller deloppgave, sammenlignes med eksisterende besvarelser i database. I databasen gjenspeiles disse gjennom relasjonene ”`Answer`” og ”`FileAnswer`” slik man ser av tabell 2.

Tabell 2 Utdrag av besvarelser lagret i database

```
mysql> SELECT answer.answer_id, itemname, subjectname, zipfileanswername
    -> FROM answer join fileanswer ON (fileanswer.answer_id =
answer.answer_id);
+---------+----------+-------------------------+----------------------+
| id      | item     | subjekt                 |besvarelse            |
+---------+----------+-------------------------+----------------------+
|      63 | item15   | SubjektP3               | srcA.zip             |
|      64 | item15   | SubjectP3               | srcB.zip             |
|      65 | item15   | SubjectP3               | srcC.zip             |
|      77 | item17   | SubjectP1               | srcA.zip             |
|      78 | item17   | SubjectP2               | srcA.zip             |
|      79 | item17   | SubjectP3               | srcA.zip             |
```

Som man ser av tabell 2, finner man igjen besvarelser for deloppgaver i databasen når man utfører en `join` operasjon på relasjonene `Answer` og `FileAnswer`. `Join` operasjon betyr at en nøkkelverdi må finnes i begge relasjonene, i dette tilfellet `answer_id.`

Forenklet uttrykk[31]: `^(.*)(\d+|\w+).zip`

Koden over viser et eksempel på et regulært uttrykk som kan hente ut informasjon om delsvar fra et filnavn. De to gruppene kan kun håndtere situasjoner hvor informasjonen legges sist i filnavnet. Skal man ta høyde for at nummeret kan innlemmes i, trenger man litt sterkere uttrykk.

[31] Forklaring: \d betyr et siffer fra null til ni. For å få en bokstav i område a-z og A-Z benyttes \w. Parentesene benyttes for gruppering. Skal et tegn forekomme null eller flere ganger benyttes ”*” tegnet, mens skal det forekomme minst en gang benyttes ”+”. Tegnet ”?” har flere betydninger avhengig av plassering. ”*?” gjør at man avgrenser søket etter nye tegn. Den vanlige betydningen er at et tegn forekommer null eller en gang. For mer komplett liste, sjekk med Javas dokumentasjon,

Utvidelse 1: `^(.*)(\(?\d+\)?|\w+).zip`
Utvidelse 2: `^(.*)(\(?\d+\)?|\w+).zip\(?\d*\)?`

Skal man i tillegg ha ulike versjoner, slik som formatene ”`Item-subjekt(nummer).zip`” og ”`item-subject(nummer).zip(kopi)`” illustrerer, må man utvide uttrykkene enda mer, slik vist i utvidelse 1 hvor ”`\d`” er endret til ”`\(?\d+\)?`”. Ved å gjøre tilsvarende på slutten kan man også ha ulike kopier. Når man kopierer filer på filsystem kommer gjerne ordene ”`Kopi`” eller ”`Copy`” med. Uttrykkene ovenfor er ikke sterke nok til å ta høyde for disse ordene. Ser man på hvordan mønsteret av kompleksitet, ser man at det er umulig å ta høyde for alle kombinasjoner som kan forekomme fordi det finnes et uendelig antall kombinasjoner å ta av. Filnavnene bør formaliseres for at denne metoden skal fungere.

I noen sammenhenger trenger man opprensking av data Ved å utføre en renskeoperasjon slettes gamle tabeller før de opprettes på nytt og man er klar til å legge inn besvarelsene på nytt. En slik opprenskning oppretter tabellene på nytt. I denne rapporten går jeg ikke nærmere inn på problemstillinger rundt opprettelser av tabeller via Hibernate[32] fordi dette er innordnet i andres oppgaver, men som benyttes indirekte for at de data man skal legge inn blir tatt vare på.

5.1.2 Tidsforbruk:

Datastrukturene som inneholder tidsforbruk fra front end SESE kan komme i form av databasefiler eller regneark. Begge kan oppføre seg som databaser, og av den grunn står valget mellom ”Java Data Base Connection (JDBC)”, og Hibernate[33] for henting av disse dataene. Filene som inneholder tidsforbruk data i fase 2 designes som en rene tekstfiler som inneholder antall minutter brukt for en oppgave eller deloppgave. Hvordan skal så disse filene knyttes til de konkrete besvarelsene de gjelder? Identifiseringen skjer gjennom filnavnet som består av tekststrengen ”`effort`” etterfulgt av navnet på besvarelsen slik det vises i fig. 12.

[32] Hibernate: se http://hibernate.org/ for mer informasjon.

[33] Diskusjon rundt JDBC og Hibernate er ikke interessant for denne oppgaven, men vil heller henvise til Sørlie(2007) for diskusjon rundt metoder for persistens.

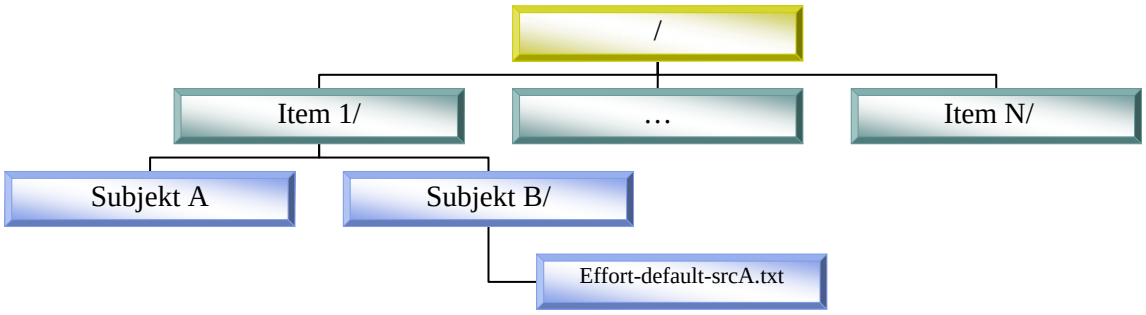


Fig. 12 Tidsforbruk

Fig. 12 viser et eksempel på hvordan tidsforbruk knyttes opp mot en standard karaktersetter for en besvarelse av deloppgave ”A” i Item 1 levert av subjekt ”B”. Dette er den besvarelsen man finner igjen som ”`srcA.zip`” i fig. 10 og 11. Grunnen til dette er at det, i tilknytning til plassering på katalogstruktur, er enkelt å identifisere hva dataene gjelder og hvilke besvarelser de tilhører, I implementasjonen benyttes det en fil for å angi hvilke felt som gjelder de ulike delsvarene, slik at man kan hente ytterligere informasjon om tidsforbruk knyttet til deloppgaver. Slike transformasjonsdata kan følge et format som det vises et eksempel på i tabell 3.

Tabell 3 Kobling av data mellom SESE og JCAT

SESE	JCAT	INFO
Totaltid	Totaltid	
P1_Tid	Lesetid	
Subjekt	Subjekt	STR_3_FIRST
P2_Tid	Deloppgave A	
P3_Tid	Deloppgave B	

5.1.3 Rapportering av skår gitt av ulike karaktersettere

Generelle tabeller defineres gjennom at de består av et variabelt antall celler og hvor cellene i den øverste raden kan utgjøre en header, men dette er valgfritt. Man kan implementere slike tabeller på ulike måter. Ved å gjøre den fleksibel i begge akser kan man legge til nye felter, slik som karaktersetter, item og/eller subjekt, og hvor de indre cellene inneholder skår.

I figur 5 så man ulike måter å lage slike tabellformater. For å forenkle problemstillingen er det fordel å lage en tabell per karaktersetter slik man ser i tabell 4.

Tabell 4 Tabellformat for en standard karaktersetter.

	item14
Subjekt 1	0
Subjekt 2	0
Subjekt 3	0

Ved å endre litt på formatet og å legge til en kolonne ”Grader” (karaktersetter) får man følgende resultat som vist i tabell 5.

Tabell 5 Tabellformat med karaktersetter

		item14
Subjekt 1	Automatisk Skåring	0
Subjekt 2	Automatisk Skåring	0
Subjekt 3	Automatisk Skåring	0

Dersom man sammenligner overstående tabell med tab 5. ser man at dersom man legger til flere karaktersettere, vil man etter hvert få et mønster som ligner litt på den tabellen man finner i øverste venstre hjørne. Tabellene er begge generert gjennom metoder definert i en implementert bruker ”Administrator” som generer innholdet i tabellene, samt en tabellprinter som håndterer formatet på disse. Ved å gjøre nytte av at alle tabeller kan settes sammen av rader og kolonner, kan disse lett utvides, dersom man ser bort i fra begrensninger i bredde eller høyde.

5.2 Evaluering og skåring av besvarelser

5.2.1 Implementering av evaluering av pakkede besvarelser

Det finnes ulike typer filer som man i utgangspunktet kunne tenke seg å behandle i et rammeverk, men som jeg nevnte i innledningen, er det kun filer som innholder Java kode som er av interesse. Dersom man har mange filer som inngår i besvarelsen, vil disse sannsynligvis være pakket sammen front end. Fordi man er interessert i kildekoden er ikke ”JAR” -formatet egnet å benytte som utgangspunkt. Derimot er filer på ”zip” format godt egnet. SESE kan fra før eksportere filer på zip format. Komprimerte filer, slik som ”zip”-filer, har en innholdsfortegnelse. Denne innholdsfortegnelsen gir så muligheten for å få tak i innholdet av de ulike elementene. Innholdet definert av denne listen må så pakkes opp og plasseres på et område på disken. Etter at disse er pakket opp, må pakkedeklarasjonene kontrolleres og eventuelt endres i henhold til den struktur de skal følge for at besvarelsen skal kunne evalueres og skåres. Slike endringer må også utføres i

importeringer av disse pakkene. Før evaluering og skåring kan utføres, må en rekompilering av besvarelsen kjøres. For å gjenkjenne pakkenavn, gjøres en sammenligning med katalogplasseringen. Man vet at pakkenavnet skal stemme overens med navnet på katalogen hvor Java-filen ligger plassert. Det finnes ulike metoder for å finne dette navnet ut fra en tekststreng, blant annet med `String` sin `indexOf`-metode, men man kan her også gjøre nytte av egenskaper ved regulære uttrykk.

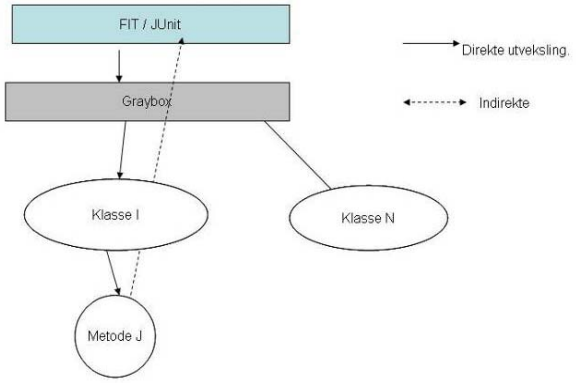
5.2.2 Implementering av Graybox evaluering

Graybox (gråboks) evaluering er en form for evaluering hvor man vet har tilgang til blant annet struktur og algoritmer[34], men kan utføre testene på svartboksnivået. Men den er ikke like gjennomsiktig som ved glassbokstesting, hvor man direkte kan teste de enkelte komponentene. En enkel svartboks evaluering kan utføres gjennom et kommandolinjemiljø.

Figur 13 Skjematisk plassering av Graybox mellom testcase og besvarelse.

Som det framgår av fig. 13, er Graybox tenkt som et mellomlag mellom testcase og besvarelsene som skal evalueres og skåres. Grunnen er at en del tester i JCAT har vist seg å reagere på kompileringsfeil, noe jeg har snakket litt om i kap. 4.2. Graybox sørger for tilgang til struktur i besvarelse, men som samtidig sørger for at test av besvarelser utføres på svartboksnivå. Den skal også ta høyde for at plasseringen kan forandres, slik at den må referere til klasser og metoder gjennom tekstlige beskrivelser av disse.

[34] http://en.wikipedia.org/wiki/Software_testing.



Figur 14 Kall til metoder gjennom Graybox.

Figur 14 viser hvordan man, fra testcase kan gå via Graybox for å kalle kode som ligger i en besvarelse, representert ved hvite sirkler. En utfordring er hvordan man skal lage en instans av en klasse man ikke kjenner på kompileringsstadiet. For å gjøre dette kan ikke det reserverte ordet ”new” benyttes, slik man vanligvis gjør. Gjennom å benytte Javas klasselaster og metoden ”`newInstance`” dette problemet unngås. For at en metode skal kunne utføres må den korrekte metoden søkes etter, for deretter å påkalle den (”`invoke`”[35]). Koden nedenfor viser hvordan bruk av Javas klasselaster (”`class loader`”) kan laste og lage instans av klasser, på tross av at klassenavnet ikke er kjent under kompilering av rammeverket. Klassenavnet er gitt gjennom en variabel av typen ”`java.lang.String`” som heter ”`evalClassName`”.

```
try {
/* load class */
 this.evalClass = ClassLoader.getSystemClassLoader().loadClass(
     evalClassName);

 /* create new instance of class, equiv. to 'new', but
 * because class isn't known during compilation,
 * the new operator can't be used.
 * */
 this.evalClassInstance = evalClass.newInstance();
} catch (InstantiationException ie) {
 System.err.println("Could not create instance of "
   + evalClass.getName() + "\n" + ie);
} catch (IllegalAccessException iae) {
 System.err.println("Could not create instance of "
   + evalClass.getName() + "\n" + iae);
} catch (ClassNotFoundException cnf) {
 System.err.println("Class not found " + cnf);
}
}
```

[35] Invoke: Påkalle eller å mane frem (kilde: Engelsk - Norsk ordbok)

```
try {
 new GrayboxEval("item16.Solution", "send", args, null).execute();
} catch (Exception e) {
 try {
   new GrayboxEval("item16.solution.Solution", "send", args,
   null).execute();
 } catch (Exception e1) {}
}
```

Slik man ser av de overstående kode, vil man i dette eksempelet ha to forsøk på å finne rett klasse og utføre metoden ”send”. Dette kan modifiseres, slik at man leter gjennom katalogstrukturen for å finne den klassen som passer nærmest. I design av slike søk kan man også benytte seg av ”Levenshtein”[36], eller regulære uttrykk for å tillate avvik i klasse og metodenavn. Levenshtein algoritmen måler avstanden mellom tegnene som inngår i to tekststrenger. I tillegg til at den tar høyde for erstatninger, måler den også metrisk distanse i forhold til innsettelser og sletting av tegn. I JCAT benyttes Similarity Metric Library[37]. Når man skal lete etter metodenavn må man vanligvis sjekke metodesignaturer. I den forbindelse trenger man å håndtere konvertering mellom objekt og primitiv.

5.2.3 Implementering av enhetstestevaluering

Enhetstestevaluering baserer på å evaluere utfallet av et sett med enhetstester for Java. Denne implementeringen kan til en viss grad utføre testene sekvensielt automatisk og hente ut de ulike verdiene som testene resulterer i. Gjennom omfattende testing av implementert kode, kan enhetstester avdekke eventuelle mangler ved denne koden. Ved å trekke ut informasjon om antall tester, feil og lignende, kan man bruke denne informasjonen til videre skåring. For å unngå konflikter med annen kode trenger man å legge disse testene på eget område. Deretter må hver enkelt testcase utføres og resultatene legges sammen. Ved omfattende tester kan denne prosessen være tidkrevende. Men det avhenger av hvor mange tester som skal utføres, og hvor store og komplekse de er. Det er to ulike formater resultatene kan komme på:

Format 1 opptrer kun dersom alle tester er korrekte:

```
C:\JCAT\tasks\src\task34solution>java -classpath c:\jcat\lib\junit.jar;.
junit.textui.TestRunner crossword.CrosswordTestCase
.....................................
Time: 0,03

OK (37 tests)
```

[36] Kilde: http://en.wikipedia.org/wiki/Levenshtein_distance

[37] http://sourceforge.net/projects/simmetrics/ (Sheffield University, UK)

Format 2 inntreffer dersom minst en av testene feiler, slik vist nedenfor:

```
C:\JCAT\tasks\src\task34solution>java -classpath c:\jcat\lib\junit.jar;.
junit.textui.TestRunner crossword.CrosswordTestCase
..F.................................
Time: 0,03
There was 1 failure:
1)
testIllegalSize(crossword.CrosswordTestCase)junit.framework.AssertionFaile
dError at
crossword.CrosswordTestCase.testIllegalSize(CrosswordTestCase.java:72
)
  at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
  at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
  at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)

FAILURES!!!
Tests run: 37, Failures: 1, Errors: 0
```

I både format 1 og format 2 er det kun de siste linjene man er interessert i. For format 1 er dette *"*`OK (37 tests)`*"*, og for format 2*:* "`Tests run: 37, Failures: 1, Errors: 0`".

Etter at enhetstestingen er utført, kan resultatene av disse testene hentes ved hjelp av regulære uttrykk. For å lokalisere testcases gjøres det først en sjekk på filnavn, deretter sjekkes det på om klassen arver egenskaper fra testcase. Dette åpner for mange muligheter. Blant annet kan man kombinere dette med metoder som sjekker for likheter, og bruke den metoden på deler av besvarelsen. Dersom man i tillegg gjør glassboks evaluering eller evaluering via Graybox, kan man gjøre tester på ulike biter av besvarelsen.

Det finnes diverse metoder for å håndtere tvetydigheter i metodesignaturer. Med tvetydigheter menes det i dette tilfellet at metodenavnene er identiske men har ulike parametre og argumenter. Denne løsningen ser kun på metodenavnet, men en forbedring vil her være også å sammenligne signaturene. Man kan også benytte seg av Levenshtein, for å finne en metodesignatur som passer best. En ytterligere forbedring kunne ha vært å bruke samme metoder som FIT benytter seg av når klasser og metoder skal lastes inn og utføres. Ulempen med det siste alternativet vil være at man gir fra seg noe av kontrollen man har. For å teste implementasjonen av skåring basert på en enhetstestevaluering, ble det laget følgende test basert på item 36:

```
/*
 * calculate score TODO: Check JUnitSubScore, JUnitScore..
*/
 public void testScoreItem36() throws Exception {
  Answer answer = null;
  try {
   answer = AnswerBank.Answers.get("item36", "solution");
   ((FileAnswer) answer).setUnzipFlat(false);
  } catch (PersistenceProviderException ppe) {
  }
  Eval junitEval36 = new JUnitEval("item36");
  // 2: SubScoring; first create a SubScoringRubric and a SubScore

  double[] subScoringRubric36 = { 1.0, 0.0, 0.0, 0.0 };

  AutomaticSubScoring subScoring36 = new AutomaticSubScoring(junitEval36,
    subScoringRubric36);

  // 3: Create a Scoring object and add SubScoring to the appropriate
  // value system in Scoring
  AutomaticScoring scoring36 = new AutomaticScoring();
  scoring36.setRegressionSubScore(subScoring36);

  Item item36 = new FileItem("Item36", "Geometry Calculator", scoring36);
  double dblScore = new Experiment().submitAnswer(item36, answer);
  int score = item36.getIntScore(answer);
  assertEquals(19.0, dblScore, 0); //Currently returns 14.0
  assertEquals(19, score, 0);

 }


Test case match:: public class GeometricCalculatorTestCase extends
TestCase {
Filename item36junit.testJCAT.GeometricCalculatorTestCase
calculateResult(...): ..............Time: 0OK (14 tests)
```

5.2.4 Differanse evaluering

Koden som genererer selve rapportene er skrevet av Sindre Mehus. Ved å lage et grensesnitt mot denne koden, kan man gjenta prosessen med å generere differanserapporter mer effektivt. Man behøver også kun å definere en referansebesvarelse for hvert item. I tillegg til automatiske rutiner for evaluering vil det også være nyttig med halvautomatiske metoder for å sammenlikne differenser mellom original og modifisert kode. For eksempel en menneskelig karaktersetter vil ofte vite hvilke endringer en person har utført i den originale koden. Til dette kan man ta i bruk et differanseverktøy som kjøres ved initiering av en besvarelse etter at den er innhentet. For å kunne gjøre en sammenligning av besvarelser, må man ha en referanse. Referansen blir definert gjennom innholdet i en tekstfil. Denne filen inneholder navn på oppgave, navn på subjekt og besvarelse som inngår i referansen. Etter at en rapport er generert må den manuelt gjenomgås av en person for kontroll. Denne jobben kan ikke gjøres automatisk, men man kan lage grensesnitt hvor vedkommende kontrollør kan registrere resultatet av sin analyse. Denne metoden kan gjentas automatisk for alle besvarelser og alle item, gitt at referansene er angitt.

Algoritmen som benyttes i denne prosessen beskrives som følger:

Trinn 1. Lokaliser referansebesvarelse og kopier denne.

Trinn 2. Initiering besvarelse.

Trinn 3. Kompilering av besvarelser.

Trinn 4. Utfør differanserapportering.

5.2.5 Vurderinger av delsvar

Til nå har vi sett på hvordan evaluering kan anvendes på enkeltbesvarelser. I dette avsnittet ser man på de tilfellene hvor flere besvarelser henger statistisk sett sammen. Med dette menes at man kan ikke se resultatene som uavhengige i forhold til hverandre. I de tidligere tilfellene er spørsmålene uavhengige av hverandre, men nå må vi se på de tilfellene hvor spørsmålene bygger på tidligere spørsmål. Organiseringen av delsvar, er gjort slik det er vist i avsnitt 5.1.1.

Når man skal gi en total skår på slike besvarelser har man tre tilnærminger til mulige design. (1) Man definerer et item for hver deloppgave som så grupperes sammen til et felles item. (2) Man ser på de enkelte besvarelsene som deler av en større besvarelse, eller (3) man definerer en gruppe kriterier, og gjennom et regelsett beskriver hvordan den endelige skåren skal beregnes ut fra disse. Et slikt regelsett kan vektlegge ulike egenskaper ved ulike deloppgavene, og gi en sluttskår basert på hvor mange av oppgavene som er fullført i henhold til de krav som er gitt. Dette er til sammenligning ikke så ulikt hvordan skåring allerede utføres, men forskjellen ligger i at man utvider skåringen til å omfatte flere svar i stedet for ett. De tre ulike alternativene ser på delsvar fra ulike perspektiver. Løsning (1) knytter et delsvar til en egen deloppgave med egne evaluerings og skåringsmetoder. Dette kompliserer problemstillinger rundt oppgaver, men likner mer den reelle forholdet mellom delsvar og deloppgave. Løsning (2), som grupperer sammen besvarelsene, tar ikke tilstrekkelig hensyn til forskjeller som kan eksistere mellom de ulike deloppgavene. Vurderinger av deloppgaver (”Sub Task Scoring”) starter fra skåringsperspektivet og setter sammen et sett av ulike delskåringer for deloppgaver (”Sub Task Subscoring”). Hver av delskåringene kan utføre et sett av evalueringer og skåringer. En metode for å hente delsvar, vises i vedlegg A.

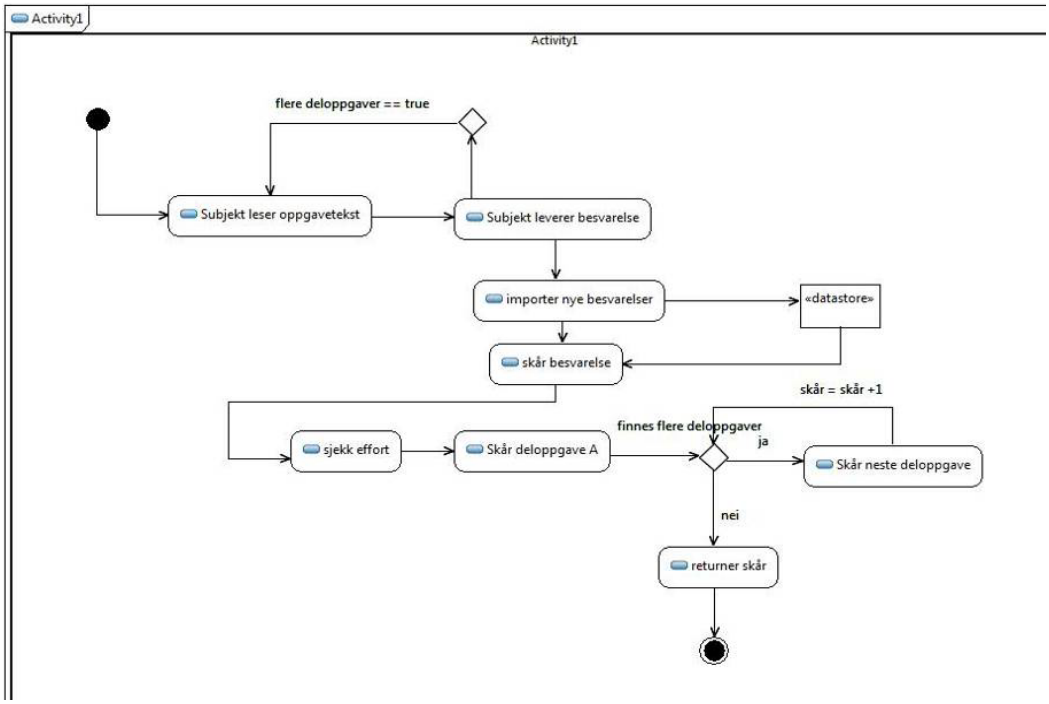


Fig. 15 Hendelsesforløp for skåring av delsvar.

Fig. 15 viser prosessen for hvordan delsvar kan evalueres og skåres. Man starter med at subjektet leser oppgavetekst og leverer inn ett eller flere delsvar. Etter at disse mottas via front end og hentes inn i rammeverket, ønsker man å utføre en skåring av besvarelsene. I dette tilfellet starter man med å vurdere tidsforbruket opp mot kriteriene som gjelder for oppgaven. Dersom disse kriteriene tilfredstilles, utfører man skåring og evaluering av siste delsvar. Om kravene for siste deloppgave ikke tilfredstilles fullt ut, er det ønskelig at man isteden ser på forrige deloppgave hvor prosessen med evaluering og skåring gjentas. Hvis ikke, vil den avslutte og returnere den skåren den har kommet frem til første delsvar. Denne prosessen er omvendt av den man finner i høyre halvdel av figur 6.

Delsvar håndterers på samme måte som vanlige besvarelser, bortsett fra at de må grupperes sammen når den totale vurderingen skal utføres. Hvert av delsvarene vil bli behandlet som selvstendige svar ved de individuelle evalueringer, men resultatet blir håndtert annerledes når spørsmålene bygger på hverandre (jfr. figur 6, høyre side). Evalueringen og skåringen skjer ved at siste delsvar evalueres og skåres først. Deretter traverseres disse bakover, og til slutt returneres den høyeste skåren som er beregnet. Dersom et subjekt sender inn flere besvarelser etter hverandre, vil kun den eksisterende besvarelsen bli gitt skår. Hvis man erstatter den eksisterende besvarelsen med en ny, vil man kunne gi skår på den nye besvarelsen, men da går den opprinnelige besvarelsen tapt. En måte å unngå dette på er å benytte nummerering av besvarelser, slik at disse gis et sekvensielt nummer etter hvert som besvarelser mottas. Dette kan tillate at man kan operere med flere versjoner av besvarelsene.

5.2.6 Tidsforbruk for besvarelser av hel- og deloppgaver

I dette avsnittet skal jeg se på hvordan man kan trekke inn informasjon om tidsforbruk som kan benyttes i skåring av besvarelser.

Data for tidsforbruk kan innhentes fra SESE som ulike filer og formater. I denne løsningen velges det kun et av formatene, nemlig en databasefil basert på Microsoft Access (”MDB”). I dette formatet eksisterer en tabell med oppgave, navn på oppgave item, starttid, totaltid, og tid forbrukt for de ulike underoppgavene. Det finnes også tilleggsinformasjon som inneholder resultat av spørreskjema, men dette sees ikke på i denne sammenhengen. Videre, når man også skal håndtere tidsforbruk som grunnlag for skåring, bør man skille mellom lesetid og tiden som benyttes til utviklingen av programkoden. I den totale tidsforbruket er det flere komponenter som inngår. Tiden som går med til lesing av oppgaveteksten må trekkes fra den totale tiden fordi den ikke er inngår som del av tidsforbruket til utvikling av besvarelsen.

Tidsforbruket kan deles inn i flere målinger: Man kan måle det totale tidsforbruket, gjennomsnittlig tidsforbruk, eller tidsforbruk per underoppgave. Dette gir også utfordring for hvordan man skal ta vare på denne informasjonen. Man kan også være interessert i å differensiere mellom tid forbrukt på å lese oppgaveteksten og tid forbrukt på å løse oppgaven, slik det vises i figur 16. Dersom man kun tar vare på total tidsforbruk gir det utfordringer i forhold til slike beregninger.

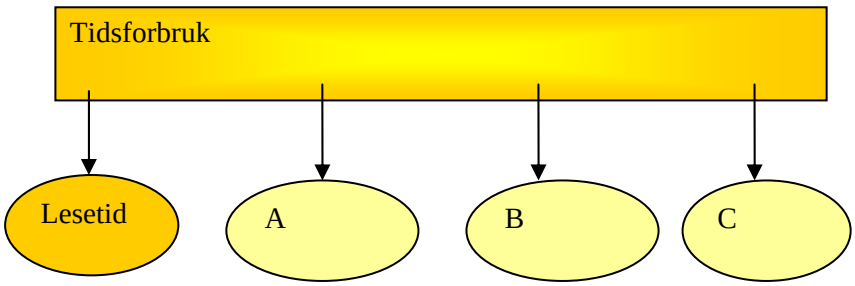


Fig. 16 Tidsforbruk for deloppgaver.

Tabell 6 Plassering av tidsforbruk i database

ID	ITEM	SUBJEKT	TIDSFORBRUK
40	item14	Subjekt 1	12
42	item14	Subjekt 2	17
44	item14	Subjekt 3	6
48	item15	S011	5
49	item15	S011	46

Tabell 6 viser hvordan tidsforbruk, knyttet til deloppgavene, plasseres i en databasetabell, men her finnes også andre alternative løsninger, slik som å lage en egen tabell for tidsforbruk.

5.2.7 Implementering av regelsett for skåring

I dette avsnittet ser jeg på hvordan man kan lage et regelsett basert på evaluerings og skåringskriterier, som man benytter blant annet i forbindelse med manuell skåring. Koden under viser hvordan man, gjennom å overføre skår for ulike deloppgaver og tidsforbruk, kan lage et regelsett som tar høyde for disse gjennom å benytte *if* setninger. Gjennom å lage en vektor med ulike karakterer, kan man angi hvilken karakter de ulike kriteriene skal gi. Denne kan lett erstattes med nye verdier uten at man endrer reglene. Reglene tillatter også avvik. Disse avvikene defineres gjennom terskelverdier for tidsforbruk og skår av delsvar. Regelsettene kan utvides til å gjelde besvarelser som strekker seg over flere oppgaver.

```
/**
* Calculate score according to re-defined rules
*/
public int calculateScoreInt(Map<String, Double> scores) {

int effort = (int)scores.get("totalEffort").intValue();
if (effort > limit + timeThreshold || scores.get("A")< maxScore -
threshold)
  return scoringVector[0];

if (scores.containsKey("B") && scores.get("B") < maxScore - threshold)
  return scoringVector[1];

if (scores.containsKey("C") && scores.get("C") < maxScore - threshold)
  return scoringVector[2];

if (scores.containsKey("effort") && scores.get("effort")< limit -
timeThreshold)
  return scoringVector[4];
else
  return scoringVector[3];
}
```

Hensikten med reglene i denne koden er å definere regler for skår tilsvarende dem som vises i tabell 7.

Tabell 7 Regler for skår på tabellform

Krav / Skår	Tidsforbruk	A	B
3	OK	OK	OK
2	For høyt	OK	OK
1	OK	OK	FEIL
0	OK	FEIL	FEIL
0	FEIL	FEIL	FEIL

Forklaringen av tabell 7 er at for å få full skår i følge denne tabellen kreves det at tidsforbruket ligger innenfor gitte krav til total tid og lesetid, og at besvarelser for A og B begge er korrekte. Er tidsforbruket for høyt, men begge delsvar fungerer, kan man redusere karakteren. Dersom A fungerer men ikke B, trekkes ytterligere en karakter fra. Uansett om man forholder seg innefor tidrammene eller ikke; Dersom begge besvarelsene er gale, får man karakteren ”0”.

Algoritmen som benyttes for denne typen evaluering og skåring er som følgende:

1. For hver deloppgave: definer testcase og evalueringsmetode.
2. For hver evalueringsmetode: definer skåringsmetode.
3. Lag regelsett, hvor regel *[i := 0 til N] < (krav[i] – δ (terskel)).*
4. Sett nytt verdisystem.

Ser man nærmere på reglene, kan skår for A, B, og C lett erstattes med andre regler. Dette gjør det enkelt å omdefinere reglene til å gjelde andre egenskaper.

5.2.8 Karaktersettere

Besvarelser, enten de er små eller store, enkle eller sammensatte, trenger å bli karaktersatt for at man skal kunne sammenligne disse med andre besvarelser for samme oppgave.

Prosessen en manuell karaktersetter må utføre for å karaktersette en besvarelse er ikke så ulik den som utføres automatisk. Det første som må gjøres, er at besvarelsen må åpnes, kompileres og kjøres.

Deretter kan utskrifter og eventuelle grafiske grensesnitt vurderes i henhold til kriterier som gis for oppgaven. Avhengig av hvor mange av disse kriteriene som oppfylles, gis det en karakter. En karakter gis på bakgrunn av en eller flere individuelle delskår, som hver for seg kan beregnes ut fra ulike kriterier. Man kan legge skårsammenligne karakterer fra menneskelige karaktersettere med automatiske, kan man få mer informasjon om avvik i skåringen, og på bakgrunn av disse eventuelt korrigere kriteriene som ligger til grunn. Resultatene fra disse må kunne tas inn i rammeverket for å få bedre sammenligningsgrunnlag, men også fordi det i visse sammenhenger finnes besvarelser som ikke kan skåres på andre måter. I enkelte situasjoner er det praktisk å kunne sammenligne innholdet i besvarelsen mot en fasit. Som støtte til dette kan de i tillegg til å studere utskrifter og kjøring av besvarelser, også få utskrifter fra differanserapporter som viser forskjeller grafisk mellom en referansebesvarelse og den besvarelsen man studerer. Karakterene må kunne tas inn i rammeverket, slik at disse kan eventuelt sammenlignes mot automatiske skåringer eller mot andre manuelle karaktersettere. Dersom disse ikke tas inn i rammeverket gjennom automatikk, vil dette ellers være en tidkrevende prosess. Innhenting av manuelt satte karakter vises i vedlegg B.

5.3 Endret arkitektur for rammeverket JCAT.

I forbindelse med de implementasjoner som er gjort, er det gjort en del endringer i strukturen i samarbeid med Sørlie og Bergersen for å skille foretningslogikken bedre fra persistens. JCAT ikke har eget presentasjonsnivå benyttes andre metoder for dette, slik at resultater kan presenteres på en lesbar måte. Applikasjonsnivået i JCAT gjør bruk av ulike metoder for å utføre evalueringer, deriblant FIT. Data mottatt fra front end går igjennom en egen importeringsmetode beskrevet tidligere i avsnitt 5.6 før de kan vurderes på bakgrunn av ulike metoder og kriterier. Banker for item, besvarelser, subjekter og tester er innordnet i begrepet manager. All utveksling av data mot persistens går via en tilbyder av persistent lagring. I dette tilfellet snakker man om lagring mot database, og ikke mot fil selv om det også er en persistent lagringsform. For å tilnærme rammeverket til trelagsstruktur, har man skilt mellom persistens, foretningslogikk og presentasjon. Foretningslogikken kan gjøre nytte av utenforstående metoder for evalueringer.

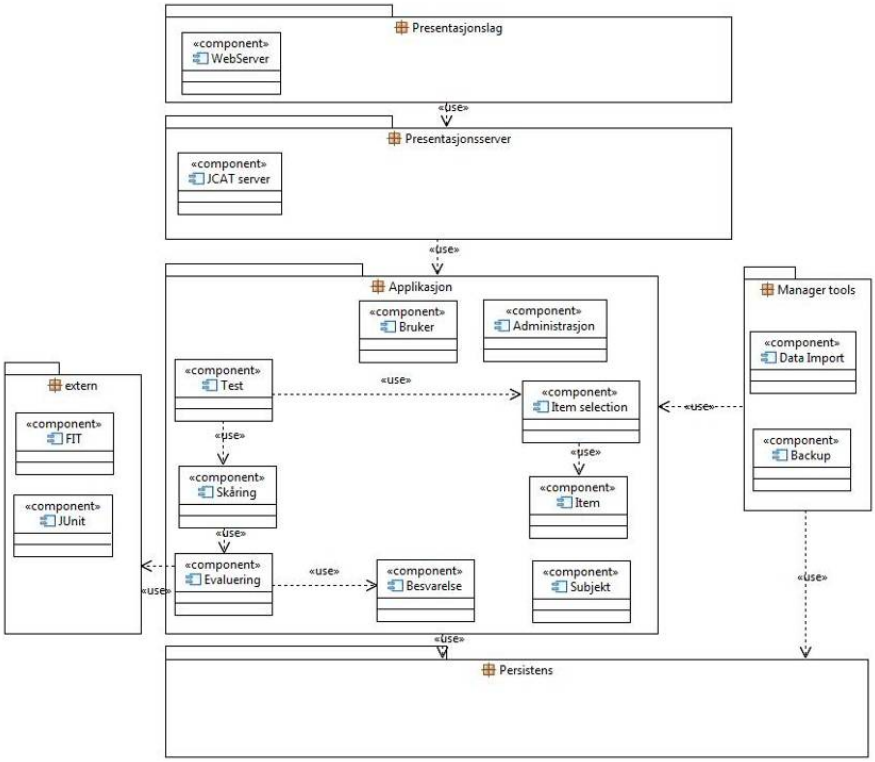


Fig. 17 Komponentbeskrivelse av rammeverket JCAT

Ved å se på de ulike komponentene som inngår i rammeverket vist i fig. 17, er det flere komponenter som inngår. På presentasjonslaget benyttes en lokal webserver som kjøres via *FitNesse* sitt rammeverk, slik at resultatene kan presenteres i en nettleser. Presentasjonsdataene hentes gjennom en presentasjonsserver, som igjen benytter komponentene som ligger i applikasjonslaget. For at disse kan få tilgang på besvarelser, må det benyttes egne verktøy for importering av data. Man kan også initiere svarbanken med ulike sett av besvarelser, tatt vare på gjennom en sikkerhetskopieringsmekanisme som er implementert. I dette rammeverket kan man gjøre nytte av eksterne rammeverk når evalueringer av oppgavene skal utføres. Det er laget en enkel brukerhåndtering, representert ved komponenten ”Bruker”. Alle administrative komponenter beskrives under komponenten *Administrasjon*. Fra tidligere har man også komponentene *Test*, *Skåring, Evaluering*, *Item Selection*, *Item* og *Subjekt*, hvor metoder for de enkelte komponentene befinner seg.

Oppsummering kap. 5: I dette kapittelet er det snakket om hvordan man skal legge til ulike typer evaluering av besvarelser. Videre er det snakket om hvordan man skal bruke slike evalueringer, når man må ta hensyn til at man spørsmål kan bygge på hverandre. Alle besvarelsene må hentes inn fra front end og organiseres. Det er sett på metoder for hvordan disse kan tas inn i rammeverket for evaluering og skåring. Besvarelser kan identifisere deloppgave gjennom navngiving eller nummerering, og man kan bruke denne egenskapen når man skal lage regelsett for å skåre delsvar. Skår satt av ulike karaktersettere kan presenteres i forskjellige tabeller, og deretter sammenlignes.

6. Diskusjon

I kap. 6.1 diskuteres valg av verktøy. Deretter analyseres resultater i forbindelse med administrasjon og evalueringer i kap. 6.2. I avsnitt 6. 3 diskuteres nærmere noen alternative løsninger. Avsnitt 6.4 ser hvordan resultatene kan ses i sammenheng med å forberede et rammeverk for computer-adaptiv testing, Til slutt ser jeg på ulike erfaringer som jeg har fått, og bruker disse i sammenheng med å evaluere eget arbeid.

6.1 Valg av verktøy

Før jeg diskuterer resultater og kravspesifikasjoner, vil jeg begynne med valg av verktøy som er gjort.

- Bruken av ”Rational Software Modeller" (RSM), til bruk av modellering av enkeltkomponenter var tungvint. Dessuten er versjonen jeg benyttet, en versjon som er tilgjengelig gjennom Institutt for Informatikk. Denne er meget kostbar i kommersiell sammenheng. I dette verktøyet må man designe hver enkelt klasse manuelt. På grunn av dette kan inkonsistens mellom reell kode og modell lett opptre. Man bør etterkontrollere modelleringen i forhold til koden ved jevne mellomrom på grunn av dette.

- Eclipse ble brukt til å utvikle prototypene, og det fungerte godt til dette formålet. Men fordi det er benyttet SVN (versjonshåndtering) fremfor ”Concurrent Versions System” (CVS), var det utfordringer i sammensyingen av programvare utviklet av de ulike prosjektmedlemmene. Man kunne heller ikke benytte seg av CVS metoder i Eclipse. CVS er mer spesifikt rettet mot parallell utvikling, i motsetning til SVN, og egner seg bedre når programvare skal utvikles i større prosjekter.

6.2 Analyse av resultater

6.2.1 Administrering

Slik tidligere nevnt er det to underbegreper som er knyttet til administrasjon. Det ene er innhenting av data, som er mest aktuelt i forbindelse med kalibrering av oppgaver, og det andre er organisering av disse.

Innhenting av data: Besvarelser ble i utgangspunktet kun innhentet og organisert manuelt, men med de metoder som er innført i denne oppgaven er deler av dette automatisert. Besvarelsene er hentet fra SESE, slik at innhentingsmetodene kun er testet mot slike data. Hvis man kan innhente besvarelser fra disse på en tilsvarende struktur, kan man med rimelig grunn anta at metodene for innhenting av data også kan utvides til andre ”front end”-applikasjoner. Selv om det mest effektive er å benytte seg av automatisk strukturering, er det enkelte situasjoner hvor man trenger manuell strukturering før besvarelsene ble hentet inn i rammeverket. Spesielt var det i forbindelse med å legge til informasjon om tidsforbruk som ble tilsendt i etterkant. En annen årsak er at dersom besvarelsene ikke var organisert på en måte som kan gjenkjennes automatisk, bør man manuelt omstrukturere disse for at de automatiske metodene kunne benyttes. En forbedring av denne organiseringen, er å utvide innhentingsmetodene til også å omfatte formater det ikke er tatt høyde for i denne oppgaven. Fordi man allerede i kravspesifikasjonen angav at man trenger å hente inn data i to trinn, først fra front end og deretter videre inn i rammeverket, har implementering av to-fase innhenting vært naturlig. Styrken ved denne, er at man kan kontrollere data før de benyttes i rammeverket. Selv om dette kun har vært anvendt på besvarelser og tidsforbruk, er det fleksibelt nok til at det kan også benyttes på andre data som trenges å innhentes i rammeverket. Begrensningen i dagens administrasjon, er at det finnes fremdeles deler som utføres semi-automatisk, slik som innhentingen av data fra front end. Dersom man trenger større effektivitet i administrasjonen, er dette et punkt som kan ses nærmere på. En annen potensiell flaskehals kan være å utføre importeringer av store mengder besvarelser på samme tid. Jeg mener at dagens løsning, som forsøker kun å importere begrensede data på et gitt tidspunkt, kan egne seg godt.

Organisering: Besvarelsene og annen informasjon som hentes inn fra SESE, organiseres først i en katalogstruktur, før de organiseres videre i en database. I tillegg til at man kan kontrollere disse besvarelsene for mulige feildata, gjør det også at man kan benytte de beste egenskapene ved begge mekanismene. Databaser har innebygde mekanismer for hurtige oppslag, men man kan ikke studere innholdet på samme måte som man kan med filstruktur. En annen fordel med databaseorganiserte besvarelser, er at man kan på sikt utvide databasene til å være distribuerte. Dette tillater at man lagrer besvarelsene, og/eller kjøre vurderingene samtidig på flere uavhengige maskiner, i stedet for en. En distribuering av besvarelser kan således bidra til økt effektivitet. Samtidig har de også mekanismer for å håndtere samtidighetsproblematikk i innhenting og uthenting av data. I denne oppgaven er det gjort begrensninger til en en-bruker situasjon, slik at det ikke har vært et problem her: Skal rammeverket på sikt utvides til å omfatte organisering av besvarelser i et flerbrukermiljø, vil denne utfordringen dukke opp. Dersom man skal gå over til mer online evaluering. kan man også vurdere mer automatikk i overføringer av besvarelser og data fra front end til rammeverket, Grunnen til dette, er at metodene som benyttes i denne overføringen baserer seg på delvis manuell uthenting av data fra dette rammeverket, og kan utgjøre en flaskehals når store mengder besvarelser skal hentes inn. Ser

man dette i sammenheng med utførelsen av tester, er det knyttet en usikkerhet i om tiden det tar å utføre alle tester, kan forbedres ved å effektivisere organiseringen noe. Så langt er det ikke gjort direkte undersøkelser på akkurat dette, og det kan dermed ikke trekkes noen slutninger på dette punktet. En annen begrensning ved dagens administrering av besvarelser, er at man kun kan operere med en versjon per besvarelse eller har noen annen form for besvarelseshistorikk. Slik historikk kan være nyttig for å se hvordan subjektene presterer på de ulike oppgavene over tid. Organiseringen av besvarelsene gjør at man enkelt kan undersøke besvarelsene, både manuelt og semi-automatisk. Det er lett å finne frem til besvarelsene som er gitt av ulike subjekter for de ulike oppgavene, og hastighetene ved oppslag er så langt ikke funnet å være et problem.

Tid: At besvarelser og tidsforbruk kan knyttes sammen på ulike måter, ser man blant annet gjennom data hentet fra SESE. Fordi andre formater kan knytte disse sammen på andre måter, kan dette gi grunnlag for å utvide eller endre metodene som benyttes i denne oppgaven. Fordi tidsforbruket måles i SESE gjennom å ta tiden som går mellom hver gang subjektet går videre i oppgaven, selv om oppgaveteksten hentes ned kun en gang. Men fordi testen ikke kan gjøres i et kontrollert miljø, kan det oppstå flere kilder til feilmålinger. Blant annet vet man ikke om besvarelsene leveres i henhold til oppgavespørsmålene, eller om de leveres etter at alle spørsmål og deloppgaver er løst. Ved å innhente data om tidsforbruk i to faser, kan det korrigere til en viss grad for feilmålinger av tidsforbruk som oppstår når et subjekt sender inn flere delsvar på samme tid, men det gis ingen garanti for dette. Dersom man ser tidsforbruket til et delsvar i sammenheng med tidsforbruket for de andre, kan man analysere tidsforbruket med statistiske metoder. Dette kan gi ytterligere informasjon om hvordan fordelingen av tidsforbruket er, og kan også bidra til mer korrekt vurdering. Per dags dato gjøres ikke slike statistiske analyser av tidsforbruket, men de kan påvirke skåringsresultater og potensielt korrigere for enkelte avvik.

Utvidbarhet: Slik man ser blant andre rammeverk, er nettopp utvidbarhet gjennom modulbasert oppbygging viktig. Dersom man ønsker å legge til ny funksjonalitet, bør eksisterende rammeverk, ideelt sett, påvirkes i minst mulig grad av disse endringene. At det unngås for mange koblinger mellom ulike moduler, øker også vedlikeholdbarheten til rammeverket. Samtidig minskes sjansen for at det oppstår komplekse feil, hvor kilden er vanskelig å oppdage. I denne oppgaven knyttes avhengigheter mellom moduler når man importerer fra andre moduler. Man kan gjøre dette mer fleksibelt ved å innsette slike avhengigheter i kjøretid, men dette krever noen endringer i forhold til hvordan rammeverket struktureres. Ved å gjøre disse knytningene når rammeverket kjøres, slipper man å stoppe rammeverket hver gang man bytter ut en komponent med en annen. Resultatene i denne oppgaven viser at man kan gjøre slike moduler uavhengige med hverandre dersom man lager grensesnitt mellom metodene. Metodene som ble laget for å definere regelsett, kan utvides på flere områder. I kravspesifikasjonen var det kun tatt høyde for at de trengtes å kunne anvendes på å lage

regler for hvordan delsvar skal skåres, gitt at man kan vurdere besvarelsene individuelt eller i sammenheng. Rammeverket er også utvidbart i forhold til nye evalueringer av besvarelser. Grunnen til at man trenger denne utvidbarheten, er at man kan få flere metoder på et senere tidspunkt som også skal inngå i vurderingen. Dersom rammeverket ikke gjøres fleksibelt nok på dette punktet, kan det oppstå problemer når de nye metodene skal benyttes.

Skille mellom evaluering og skåring: Måten man skiller begrepene evaluering og skåring på i JCAT, bidrar til at man får denne fleksibiliteten. Dersom man ser hvordan skåring av besvarelsene er implementert, ser man at disse er godt adskilte fra de evalueringer som er verdinøytrale. Dette gjelder ikke bare skåringene for hele besvarelser, men også for delsvar. Dersom man trenger å skåre besvarelser på nytt, gitt at det legges til kun en ny evalueringsmetode, behøver man kun å utføre den nye evalueringen. En annen grunn, er at man ikke behøver å endre skåringene, selv om man endrer eller legger til en ny evaluering. Skårreglene vil deretter kunne utføres effektivt på bakgrunn av denne evalueringen. Fordi man lagrer resultatene av evalueringene persistent, kan kalkuleringen av nye skår skje mer effektivt.

Sammenligning av karaktersettere: Et av kravene var mulighetene for å kunne sammenligne karaktersettere. Dette fordi man trenger å se hvordan karaktersettere med ulike kriterier og verdisystemer vurderer de samme besvarelsene. Selv om det kun fokuseres på et par alternative måter å representere disse sammenlignbare resultatene, ser man i kravspesifikasjonene at det finnes også andre måter. Fordi metodene som benyttes til å generere tabellene er uavhengige av presentasjonene, kan man bare endre innholdet i disse til også å omfatte andre data som er nødvendige for å få et sammenligningsgrunnlag. I løsningene ble det fokusert på hvordan det kunne lages individuelle rapporter for de ulike oppgavene. Dette ble gjort for å forenkle lesbarheten, men man bør se hvordan disse kan settes sammen på en måte, slik at man lettere kan sammenligne karaktersettere. Dessuten ligger det en begrensning at de kun er testet ut på en standard karaktersetter, men de er enkle å utvide dette til å gjelde flere karaktersettere per oppgave. En annen løsning som ble vurdert var å bruke pivottabeller[38], men det finnes ikke så mange fungerende metoder for å implementere slike tabeller. Bruk av proprietær kode for å lage pivottabeller er unngått for ikke å bryte med det ikke-funksjonelle kravet om åpen kildekode. Metodene som er laget for å generere utskrift av skår, har en utvidbarhet i seg. Ved å endre fra standard karaktersettere til en annen, kan man generere nye karakterer basert på den nye skåringen. Dette gjør man også i stand til å kunne gjøre sammenligninger mellom ulike karaktersettere på en enkel måte. Når disse tabellene genereres, utføres en rekalulering av skår for

[38] Pivottabell: Tabeller som kan summere statistiske data. Wikipedia.org. (2008). "Pivot table." Hentet 26. mai, 2008, fra http://en.wikipedia.org/wiki/Pivot_table.

besvarelsene. Dette gir dessverre en forsinkelse, men samtidig får man oppdatert skår i henhold til kriteriene som er satt for de ulike karaktersetterne. Dersom denne ikke hadde vært utført, kan man risikere å få skår satt på bakgrunn av foreldede kriterier, og som dermed kan være misvisende.

Klargjøring for adaptiv test: Metoder for å sammenligne skår satt av ulike karaktersettere, gjør at man kan finne forskjeller mellom de ulike karaktersettere. Slik man ser av skårrapportene i denne oppgaven, kan man med dagens metode lage en ny tabell per karaktersetter per oppgave. Man kan forbedre dette på flere områder. Dersom man vet hvilke kriterier de ulike karaktersettere vektlegger, kan man finne frem til om vektleggingen eller kriteriene trenger å justeres. Fordi metodene som benyttes for automatisk skåring er kontrollert, kan man også anta at de er anvendbare i adaptive tester. Gjennom å kunne endre kriterier og legge til nye, kan man også forbedre karaktergrunnlaget, slik at man får mer reelle vurderinger av ferdighetsnivåene til subjektene. På bakgrunn av skåringsstatistikker, kan man estimere vanskelighetsgraden til oppgavene, og dermed justere rekkefølgen på oppgavene i henhold til dette. På dette området er det fremdeles mye som gjenstår. Blant annet mangler det metoder for å sette vanskelighetsgrad for de ulike oppgavene. Videre mangler det også metoder for hvordan disse skal hentes ut igjen, slik at de kan presenteres for subjektene i en justerbar rekkefølge.

6.2.2 Evaluering og skåring av besvarelser

Det finnes styrker og svakheter ved både manuell og automatiske vurderinger av besvarelser. Besvarelsene kan med de metoder som er benyttet i denne oppgaven, både vurderes automatisk og manuelt, avhengig av hvilke egenskaper man ønsker å vurdere. Det er flere metoder som ikke er tatt inn i denne oppgaven, og som man bør vurdere å trekke inn, slik at man kan få mer komplett vurdering av besvarelser. Noen av de metodene som ikke er benyttet, er kohesjon og kopling. Disse kan gi informasjoner om egenskaper ved besvarelsene som man ikke får med metodene som benyttes i JCAT. Sett i kombinasjon med andre metoder for å evaluere besvarelser, kan disse bidra til at man får mer korrekte vurderinger av prestasjonene, hvilket igjen bidrar til mer korrekt vurdering av ferdighetsnivåene. Fordi automatiske vurderinger, i motsetning til manuelle karaktersettere skiller mellom skåring og evaluering, må man også vurdere om designet og implementeringene oppfyller dette kravet. Dersom man ser på metodene som er laget i denne oppgaven for evalueringer av besvarelser, vil de kun produsere verdinøytrale resultater gjennom utførelsen. De gjør heller ingen rangering av besvarelsene, hvilket samstemmer med definisjon av begrepet evaluering. Det er først når man beregner skår for oppgavene at man kan utføre rangeringer, hvilket igjen oppfyller definisjonen av begrepet skåring.

Semi-automatisk/Automatisk vurderinger: Manuelle vurderinger tillates ved at vurderingene man gjør manuelt, blir knyttet til besvarelsene vurderingene gjelder. Det samme gjelder også for vurderinger som skjer semi-automatisk. Forskjellen mellom disse to, er at man i det siste tilfellet kan gjøre nytte av automatiske metoder for å få tak i informasjon, som kan bidra til å få et bedre vurderingsgrunnlag. I denne oppgaven ble det innført to nye evalueringsteknikker, hvor den ene er semi-automatisk og den andre er helautomatisk.

- **Ny semi-automatisk evaluering:** Denne oppgaven implementerte en automatisering av differanserapporter. Disse rapportene kan genereres sekvensielt på flere oppgaver og flere ulike besvarelser. Fordi det defineres kun en referanse per oppgave, kan man utføre denne på alle besvarelser som gjelder for en oppgave. Dette kan så utføres på alle oppgaver som eksisterer i systemet. Disse rapportene kan så inngå i det vurderingsgrunnlaget som benyttes av manuelle karaktersettere, som så gir sin vurdering.

- **Ny helautomatisk evaluering:** Ved å utføre enhetstester serielt, kan man utvide måten man vurderer besvarelser på automatisk. Disse kan basere seg på testcase som er predefinert eller som inngår i besvarelsen. Dersom testcase inngår i besvarelsen, kan denne også inngå i vurderingene som gjøres. Evalueringer basert på enhetstesting kan være tidkrevende, avhengig av hvor mange tester som skal gjennomføres. Det kan også være begrensninger i hvor mange av testene som blir utført.

Vurdering av delsvar: For skåring av delsvar, ble det implementert egne framgangsmåter for å skåre disse. Hvordan resultater fra de enkelte delsvar inngår i den helhetlige vurderingen, er definert gjennom å lage et sett av regler for skåring. Når man skal vurdere uavhengige besvarelser, behøver man ikke å se dem i sammenheng. Da kan man bare legge sammen resultatene for de ulike oppgavene. Men når man fikk sammenhenger mellom spørsmålene, ble det behov for metoder som kunne beskrive denne sammenhengen. Fordi de ulike delsvarene kan skilles fra hverandre, kan man vurdere hver av dem individuelt eller som del av en helhetlig oppgave. Det ble valgt å beskrive denne sammenhengen mellom disse gjennom å definere ulike sett av skåringsregler. I JCAT ble regelsett sett på som en del av et verdisystem fordi man kan benytte regler til å lage et nytt verdisystem. Reglene kan lett gjenbrukes eller tilpasses til nye kriterier. Man kan også lage ulike regler som gjelder for samme oppgave, og erstatte disse med nye dersom man trenger å endre kriteriene som ligger til grunn for skåringen. Det er vurdert to alternative rekkefølger som delsvarene kan vurderes i. Den første starter med å vurdere sist mottatte delsvar, og går bakover til tidligere mottatte delsvar, mens den andre går den motsatte veien. Fordelen med å starte på sist den innleverte besvarelsen er at, dersom denne besvarelsen er korrekt, behøver man ikke å vurdere tidligere delsvar. Fordi JCAT mangler tilstrekkelig historikkhåndtering av besvarelser, kan man benytte nummereringer av besvarelser som en tilnærming. Det fantes flere alternative løsninger som ble vurdert. En annen måte

å løse utfordringen ved å skåre delsvar på, enn den som ble valgt i oppgaven, var å gruppere sammen besvarelser. Ulempen er at man endte opp med å anvende samme regel, hvilket gav mindre fleksibilitet. Fordi man trenger å kunne anvende ulike regler på ulike deloppgaver, anså jeg altså ikke dette som en tilstrekkelig god nok løsning. Alternativ to var å lage en gruppering av oppgaver som kan defineres som underoppgaver. Dette gjør at man igjen får større fleksibilitet med å anvende ulike regler på ulike deler, men man får nye utfordringer i forhold til den sirkulære assosiasjonen som oppstår. I praksis lager man altså en kobling mellom et item og flere andre item, slik at man trenger å skille mellom selvstendige oppgaver og de som er satt sammen av flere. Den tredje løsningen, som var den som ble valgt, var å lage regelsett som grupperer sammen evaluering og underliggende skåring. For å få tilgang til de enkelte delsvarene, måtte også det første alternativet tas i bruk. Fordi man har definert ulike regelsett, kan man angi hvordan resultatene skal ses i sammenheng med hverandre. Man kan legge også til nye regler eller fjerne noen, uavhengig av data som inngår i vurderingsgrunnlaget. Det finnes også en annen styrke ved disse regelsettene, som ikke er vurdert i kravspesifikasjoner eller design: Fordi man kan definere reglene på en slik måte, at man kan ignorere de dersom man ikke har vurderingsgrunnlag for dem, kan de likevel tas i bruk på senere tidspunkt.

Uavhengighet: Ved å kombinere svartbokstesting og tilgang til struktur i besvarelsene, kan man skille testcase og de besvarelser man ønsker å vurdere. Samtidig gjør det at man får bedre testing av oppgaver som allerede er definert i JCAT. Dette gjør at man ikke er avhengige av bestemte strukturer på besvarelsene, og dermed er friere. Man sørger dermed også for at skåring og evalueringsresultatene ikke påvirkes av feil i tidligere vurderte besvarelser. Dette igjen medfører at man får mer korrekte skår. Samtidig sørger den for å innkapsle unntak som kan oppstå i besvarelsene, slik at disse kan håndteres på et mer overordnet nivå. Ved å sende ved den opprinnelige feilen som årsak, kan man likevel få informasjon hvorfor unntaket inntraff. Graybox tar utgangspunkt i ”Language Reflect”-pakken i Java, noe som gir utfordringer i forhold til implementasjon. Spesielt er disse utfordringene knyttet til å gjenkjenne de deler av besvarelsene man er interessert i å vurdere. På dette området er det flere oppgaver som trenger denne funksjonaliteten, men man kan lett anvende denne metoden på andre oppgaver i tillegg til de som benytter den i dag. Mye av det som gjøres i den metoden, finner man også i andre metoder, men fordi man trenger mer kontroll over utførelse, er det i praksis ingen andre kjente måter å gjøre det på.

6.3 Forberedelse for computer-adaptiv test

Bidragene i denne oppgaven, skal være med på å klargjøre rammeverket JCAT for computer-adaptive tester (CAT). Disse bidragene må således settes i sammenheng med denne målsettingen.

Administrasjon: Fordi man har kunnet innhente og organisere ulike former for besvarelser, kan disse besvarelsene også tas inn når rammeverket utvides med funksjonalitet for CAT. Selv om metodene fungerer ved innhenting, mangler det metoder for å utveksle item informasjon den andre veien. Dersom subjektene skal få oppgaver basert på valg foretatt av dette rammeverket, bør det også lages metoder som kan sende informasjon fra JCAT til en front end mot subjektene. Ellers blir det vanskelig å gjennomføre computer-adaptive tester. Oppgavene som sendes ut kan ha definert rekkefølge, tilfeldig rekkefølge, eller rekkefølge basert på skår fra en eller flere tidligere innleverte oppgavebesvarelser. Sammen med Linda Sørlie og Gunnar Bergersen, ble det laget et arvehierarki for ”Test”, med tilhørende eksperiment og kalibrering. Gjennom dette hierarkiet kan besvarelsene vurderes, og i tilknytning med itemutvelger, kan ny oppgave plukkes ut.

Utvidede vurderingsmetoder: Denne oppgaven har sett på hva som skal til for å legge til to nye evalueringer. Fordi nye evalueringer kan legges til på tilsvarende måte, kan man få flere måter å basere skår på. Dette medfører at man kan få et bedre grunnlag for ferdighetsestimater, og dermed bidra til bedre utvelgelser av oppgaver.

Uavhengige vurderinger: Metodene for å skille testcase og besvarelser er bedre enn direkte assosiering, fordi man unngår å måtte starte tester fra begynnelsen for å produsere korrekt resultat. Dermed øker også sannsynligheten for at kvaliteten på skår som beregnes også forbedres, selv om dette er kun en av metodene for å gjøre dette. Da kalibrering av oppgaver baserer seg på store mengder data med respektive skårer, vil denne også kunne basere seg på et bedre data grunnlag. Dette vil igjen påvirke uthentingen av de oppgavene som skal presenteres for subjektene, slik at man lettere kan plukke rett oppgave. Ettersom kalibreringen baserer seg på oppgave- og/eller subjekt statistikker, vil feil i tidligere skåringer kunne medføre mulighet for at kalibreringen blir ukorrekt. Estimatene, som gjøres på bakgrunn av disse kalibreringene, kan dermed kunne påvirkes ytterligere. Skal initialvurderingen av ferdighetsnivået være mest mulig korrekt, slik at man får et mest mulig riktig inngangspunkt for utvelgelse av første oppgave, bør naturligvis disse estimatene være mest mulig riktige. Dette stiller igjen krav til at skåringer og evalueringer av besvarelsene ikke påvirkes av ytre faktorer, og det er akkurat dette punktet som Graybox kan bidra med. Et annet viktig bidrag, er at Graybox kan forsøke flere alternativer. Dette gjør at man kan finne en fungerende løsning, selv om evalueringen av besvarelsen feiler på tidligere forsøk. Hvor mange forsøk som trengs, kan variere mellom ulike oppgaver og besvarelser. At vurderingene ikke påvirkes av ytre faktorer, hjelper ikke så mye dersom det finnes feil i datagrunnlaget. Slike feil kan lett oppstå, enten ved at subjekt leverer en besvarelser som ikke angår oppgaven, eller at et subjekt velger å sende inn alle besvarelsene etter at man er ferdig med å utføre dem. I den første situasjonen risikerer man å gi skår på bakgrunn av ukorrekte data.

Sammenligninger av skår: Ved å skille ut begrepet ”karaktersetter” i rammeverket, er det mulig å undersøke hvordan ulike karaktersettere skårer identiske besvarelser. Sammenligningene kan både foregå mellom manuelle karaktersettere og automatiske karaktersettere med ulike verdisystemer. Videre kan nye karaktersettere legges til med utvidede metoder for evaluering, slik at man kan vurdere om det er hensiktsmessig å for utvide et testskript for en oppgave fra for eksempel 10 til 1000 test tilfeller.

6.4 Ikke-funksjonelle krav

6.4.1 Plattformuavhengighet

Selv om rammeverket JCAT tar utgangspunkt i programmeringsspråket Java, som er plattformuavhengig i seg selv, behøver man ikke lete lenge i kildekoden før man finner utfordringer i forhold til plattformuavhengighet. Plattformuavhengighet er viktig dersom man på et senere tidspunkt skal overføre JCAT over til ny maskinplattform. Det er noen begrensninger i dagens implementasjon av dette rammeverket som bør påpekes: Dagens versjon benytter innhenting av enkelte operativsystemavhengige informasjoner til en viss grad. Det er likevel flere steder hvor slik informasjon med stor fordel kan benyttes, slik at rammeverket lettere kan flyttes over til ny plattform om nødvendig. En annen begrensning, er at det også benyttes hardkodede definisjoner av stier. Ved å unngå hardkoding, kan også rammeverket lettere overføres til andre områder.

6.4.2 Åpen kildekode (Open Source)

Det finnes flere eksempler på ulike standarder for åpen kildekode. Java og MySQL er noen av produktene som tar utgangspunkt i åpen kildekode. Gjennom å tilby dokumentasjon for de moduler som er implementert kan andre få innsyn og bruke denne kunnskapen til å videreutvikle eller lage ny funksjonalitet. I alle slike sammenheng, er det diskusjoner om hvor langt man skal være åpen, spesielt dersom man snakker om å utvikle applikasjoner i kommersiell sammenheng. De fleste mener at det kun betyr at kildekoden legges ved produktet. Slik man ser det fra ”Open Source Initiative” er begrep mer sammensatt. I tillegg til at kildekoden legges ved applikasjonen, er det i følge disse flere krav som må stilles for at noe skal kalles ”Open Source”. Dersom man studerer kravene, omfatter begrepet også lisensen som legges ved. Det finnes flere ulike lisenser som benyttes, og som setter ulike krav til hvordan man kan distribuere program og kode videre. De mest vanlige er ”GNU General Public License” og ”Common Public License”. For mer omfattende liste over lisenser, kan man finne dette på nettsidene til Open Source Initiative. At rammeverket kan kalles Open Source

betyr at kildekoden legges ved, samt at den ikke opererer med royalties eller er diskriminerende i noen forstand for å nevne noen av kriteriene. Man kan heller ikke sette begrensninger på teknologi eller benyttes produktspesifikk lisens. Følgende kriterier defineres for at produkter skal kalles Open Source:

1. Det skal være lov å selge aggregerte komponenter av åpen kildekode og lisensen skal heller ikke sette krav til royalties.
2. Kildekoden må være inkludert i programmet.
3. Lisensen må tillate endringer og arbeid som stammer fra det opprinnelige arbeidet, og de må kunne distribueres under samme lisens.
4. Man kan sette krav til at arbeid utledet fra det opprinnelige prosjektet går under annet navn slik at integriteten til opphavsperson opprettholdes. Lisensen må eksplisitt tillate at avledet kode distribueres.
5. Man kan ikke diskriminere mot personer eller folkegrupper.
6. Man kan ikke diskriminere mot forsøksfelt.
7. Lisensen må gjelde redistribuering, uten at ytterligere lisens kreves.
8. Lisensen kan ikke være produktspesifikk.
9. Lisensen kan ikke sette begrensninger på annen programvare.
10. Lisensen må være teknologinøytral.
(kilde: Open Source Initiative, http://www.opensource.org/docs/osd)

Fordelen med denne type produkter, er at de er lett tilgjengelige, og man kan studere hvordan de er implementert. Dette gjør at de lett kan benyttes i forskning eller i andre sammenhenger. Eksempler på lisenser kan finnes i vedlegg C.

6.4.3 Andre ikke-funksjonelle krav

Dokumentasjon / vedlikeholdbarhet: Dokumentasjonen i kildekoden er skrevet med henblikk på Java-dokumentasjon. God dokumentasjon gjør at man lett får oversikten over de oppgavene og de viktigste metodene som benyttes, slik at man senere kan vedlikeholde programvarerammeverket. Vedlikeholdbarhet er viktig ved all programvare som skal overtas av andre på et senere tidspunkt. Dette gjør at man i mest mulig har benyttet fornuftige variabelnavn og metodenavn, i tillegg til dokumentasjonen nevnt ovenfor. Testcase som er laget for å teste ut de implementerte metodene, viser også hvordan implementasjonene er tenkt benyttet. Således kan disse testcasene benyttes som instruksjon eller opplæring videre. Fordi man også har disse testene, kan også JCAT nå refaktoreres mot disse.

Utvidbarhet: Rammeverket JCAT er utvidbart, både i henhold til evalueringer, men også i forhold til ny funksjonalitet som kal legges til siden. Ved å legge til ny funksjonalitet, har man vist at rammeverket kan utvides, også på administrasjonssiden. Bruk av managers, for å håndtere eventuelle utvidelser, gjør at også vedlikeholdbarheten til rammeverket kan forbedres. En forbedring, når nye evaluerings og skåringsregler skal legges til eller endres, er å gå via en manager for å endre disse.

6.5 Erfaringer

Etter å ha sett på resultater, og hvordan disse kan ses i sammenheng med kravspesifikasjon, må arbeidet oppsummeres og evalueres. I dette avnittet vil jeg se på de erfaringer jeg har fått, både ved deltakelse i prosjekt, men også gjennom arbeidet som er gjort.

6.5.1 Prosjektarbeid

Høsten 2007 var dette prosjektet bestående av tre personer hvor hver person arbeidet med ulike områder av JCAT. I dette prosjektet benyttet vi tre teknikker: SCRUM, SMART og SVN.

- **SCRUM**: SCRUM ble stor sett benyttet da vi var tre som skulle samarbeide på prosjektet. Cirka annenhver uke hadde vi prosjektmøter, hvor man rapporterte endringer og hvor man la plan for videre utvikling. På bakgrunn av de rapporter vi kom med, ble punkter rangert. Man kunne også diskutere problemstillinger, og alternative måter å løse disse på. Etter hvert som de ble påbegynt eller avsluttet ble de markert på ulike måter, hvilket gjorde det enkelt å se hvor langt i progresjonen man var kommet. Dette har vært veldig nyttig, og det gav en positiv følelse å se alle punktene som til slutt ble markert som fullførte.

- **SMART:** Bruken av SMART dokumenter, hvor konkrete oppgaver settes opp, har vist seg å være en fordel med tanke på fremdrift, når man hele tiden jobber for å oppnå konkrete mål og kan vurdere progresjon ut fra disse. Utfordringen med SMART var å komme opp med så konkrete mål som mulig, men likevel tillate nok fleksibilitet til å kunne gjøre endringer, dersom dette viste seg å være nødvendig. Samtidig er vurderinger av tidsforbruk kun basert på estimater. Å gi nøyaktig tidsforbruk kan i noen situasjoner vise seg å være ganske utfordrende, slik at man kan risikere å måtte endre på disse. Likevel har det vist seg å være effektiv metode, for å organisere arbeidet som skal utføres

.

- **SVN:** Bruk av versjonshåndtering i prosjektarbeidet har fungert, men har også gitt utfordringer i sammensyingen av de endringer som er blitt utført, slik diskutert i valg av verktøy. Med versjonshåndteringen som er benyttet i dette prosjektet, kan man ikke garantere at de endringer man gjør og som fungerer lokalt vil fungere for de andre etter at disse har hentet ned endringene. Dette er en av de største begrensningene med denne metoden, og andre versjonsmetoder bør derfor velges når man jobber i prosjekt.

6.5.2 Eget arbeide

Med utgangspunkt i evalueringer av oppgaver, har jeg sett på hva som trengs for å kunne innhente, organisere og vurdere besvarelser av programmeringsoppgaver med ett eller flere delsvar. Denne forståelsen er så benyttet for å kunne designe og implementere løsningene som er kommet frem til. Til slutt har jeg sett på positive og negative sider ved de valg som er gjort, og satt dem opp mot alternative løsninger. Erfaringene jeg har fått gjennom dette prosjektet viser at det finnes mange utfordringer i forbindelse med evalueringer og skåringer av besvarelser. Jeg har også tatt tak i problemstillinger jeg mente var nødvendige for å kunne klargjøre JCAT for computer-adaptiv test. For å forstå mer av problemstillingene laget jeg en kravspesifikasjon, som dannet grunnlaget for de implementeringer som jeg gjorde senere. I tillegg laget jeg et utgangspunkt til en feilrettingsoppgave, og forsøkte å løse en tilsvarende oppgave for å få bedre forståelser av ulike utfordringer.

Utvikling av kravspesifikasjon: Kravspesifikasjonen som ble laget i utgangspunktet, skulle dekke flere områder, ikke bare de som er tatt opp i denne oppgaven. Den skulle også innbefatte områder som trengs for at JCAT kan utføre ulike operasjoner i forbindelse med computer-adaptiv test. Ut fra denne kravspesifikasjonen, ble det fokusert på noen problemområder, som dannet utgangspunktet for de som er tatt opp her. Disse måtte igjen spesifiseres ytterligere, og det er dette som danner den kravspesifikasjonen som er benyttet i denne oppgaven. Utfordringen med denne kravspesifikasjonen, var å identifisere alle problemstillinger og utfordringer som er tilknyttet de områdene man undersøker. Sannsynligvis er det bortimot umulig å finne alle, men man kan i hvert fall identifisere de fleste utfordringene. Et eksempel på hvor jeg begynte å implementere en løsning, før jeg forstod problemstillingen fullt ut, var løsningen for delsvar. Etter å ha prøvd å implementere løsningen, måtte jeg gå tilbake til design, og se hvilke andre alternativer jeg hadde.

Utvikling av feilrettingsoppgave: Hensikten med å utvikle en feilrettingsoppgave var å få førstehåndserfaring av de utfordringer som man stilles overfor når man skal definere slike oppgaver, men i tillegg kan oppgaven benyttes til testing av andre subjekter. Den første utfordringen som ble støtt på, var å definere hvilke problemstillinger man ønsker at subjektene skal testes mot. Feilene man skal introdusere bevisst i slike oppgaver, må være repetitive og ikke basere seg på tilfeldigheter.

Fordi problemstillingene i oppgaven er knyttet til parallellitet, har man flere mulige utgangspunkt å velge blant. Fordi det er snakk om parallelliseringer kan man risikere å få uventede resultater, noe som er ugunstig. I all vitenskapelig undersøkelse ønsker man å komme frem til resultater som kan gjentas av andre. Det betyr at dersom det at hendelse A etterfulgt av hendelse B produserer et annet resultat i forhold til at hendelse B kommer før hendelse A, og at man ikke kan garantere rekkefølgen på hendelsene, får man store utfordringer i forhold til dette. Sammenligner man dette med oppgaver hvor parallellitet ikke inngår, ser man at disse har større kontroll over rekkefølgen på hendelsene, og dermed er gjentakbare. Ved å definere flere ulike sekvensrekkefølger, kan man avdekke flere situasjoner besvarelsen potensielt ikke tar høyde for. Ulempen er at man manuelt må definere denne rekkefølgen, og kanskje ikke forsøker andre som kan vise seg å frembringe feil.

Deltakelse i test: Ved å forsøke å løse tilsvarende oppgave, fikk jeg erfaringer sett fra subjektets synsvinkel. Oppgaven som skulle løses var en oppgave hvor spørsmålene var basert på hverandre, og hvor besvarelsene sendes inn etter hvert som man løser disse. Som alle tilsvarende oppgaver, hadde denne også en gitt tidsfrist for tid til å lese oppgaven samt en total tid. Utfordringene som ble gitt, var basert på hverandre, slik at man måtte løse disse i en bestemt rekkefølge for å få besvarelsen til å fungere. For hver utfordring som ble løst skulle det sendes inn en besvarelse. En stor utfordring var at, dersom første oppgave ikke ble løst, kunne man ikke gå videre på neste oppgave, selv om de etterfølgende spørsmål var mye enklere å løse. Dette ble til tider enormt frustrerende, men den ble løst, selv om det tok meg mer tid enn planlagt. Lærdommen som kan trekkes ut fra dette, er at oppgavene bør tilpasses et nivå man kan forvente, på bakgrunn av utdannelser og erfaringer.

Generelt arbeid: Med utgangspunkt i en ”nedenfra – opp” synsvinkel, og fordi jeg har sett mye av dette arbeidet fra et teknisk perspektiv, har jeg kommet med løsninger på en del utfordringer som man kanskje ellers ikke ville ha oppdaget. Ved å benytte kunnskaper om andre fagområder prøver jeg å bidra med ideer på hvordan man kan løse ulike utfordringer i denne oppgaven. I tillegg til å bidra med det rent utviklingsmessige, laget jeg et utgangspunkt til en kravspesifikasjon i begynnelsen av prosjektet. Det er på bakgrunn av denne, samt utfordringer som er oppdaget i forbindelse med disse, at det er laget designmodeller og implementeringer. Gode modeller gjør at implementeringene blir mer riktige, men å finne frem til de beste modellene viser seg å være utfordrende i blant. Blant annet forsøkte jeg alle tre modellene som ble foreslått for å håndtere delsvar. Dette gjorde meg bedre i stand til å se hvilke negative og positive sider de ulike løsningene hadde. I det tilfellet viste det seg at den opprinnelige modellen skapte utfordringer, noe som gjorde at andre modeller også måtte undersøkes.

Kontroll / validering: For å teste implementeringene har jeg utviklet testcase for de ulike komponentene. I tillegg har jeg sjekket resultatene fra metodene opp mot de ulike rapportene som lages. For validering av metoder, er referanser med kjente verdier benyttet. Ved å sammenligne de resultater man får mot disse, er det vurdert hvorvidt disse resultatene tilsvarer de forventede resultater. Slik kontroll er viktig for å sjekke at evalueringer og skåringsregler er korrekte. Samtidig har informasjon fra logger og rapporter bidratt ytterligere til å kontrollere og validere disse metodene. Ved å uthente informasjon om forskjeller mellom ulike besvarelser, har jeg kunnet danne meg informasjon om hvor problemer har oppstått, og slik sett også bidratt til denne kontrollen.

7. Konklusjon og videre arbeide

Denne oppgaven har hatt tre målsetninger. Den første målsetningen var å forstå ulike utfordringer rundt administrasjon og evaluering av besvarelser. Dette var nødvendig for å kunne designe og implementere løsninger som kan benyttes videre i computer-adaptiv testing av besvarelser. Alle løsninger kan ha ulike alternativer, og på bakgrunn av disse, valgt ut de alternativene jeg fant mest nærliggende.

7.1 Administrasjon

7.1.1 Innlastingen av besvarelser

Når man mottar besvarelser fra ulike subjekter, kan man ikke forvente at de følger samme struktur. Dette gjør det vanskelig å lage testcase for bestemte deler av oppgaven, og man trenger også å unngå at disse forskjellene påvirker vurderingene av besvarelsene i større grad. Skal man kunne håndtere slike forskjeller, må man kunne tilpasse seg disse ulike strukturene. En annen utfordring med å laste inn besvarelser, er å få de over på en struktur som lettere lar seg vurdere, i motsetning til om alle besvarelser har ulik struktur. Det er laget en implementasjon som tar utgangspunkt i Javas klasselaster. Denne tillater at man kan angi klassene som skal lastes inn under kjøring av programmet, og ikke nødvendigvis kun under kompilering av rammeverket, slik man måtte tidligere. Gjennom å benytte denne kan ikke bare besvarelsene raskere lastes inn, men man får også en uavhengighet mellom testcase og besvarelser som er nødvendig. Denne uavhengigheten oppstår ved å anvende det øverste abstraksjonsnivået i Java som kan benyttes. Dette gjør at man på en bedre måte kan få testet besvarelsene, uten å frykte at besvarelsene kan påvirke hverandre. I motsetning til å initialisere testene med fungerende besvarelser, slik man måtte gjøre tidligere, kan nå testene utføres uten nærmere kjennskap om besvarelsenes struktur. Man kan også teste besvarelsene mer iherdig, før man anser dem som ikke-fungerende. Graybox er kun testet på enkelte besvarelser. Den må utvides, slik at den kan benyttes på testcase hvor den ikke allerede benyttes. En annen begrensning, er at den foreløpig ikke gjør flere forsøk på å finne fungerende besvarelse, men kun er avgrenset til et par forsøk. Dette kan enkelt korrigeres for, slik at flere forsøk tillates.

7.1.2 Innhenting av data fra front end

SESE er et av flere front end som kan benyttes mot subjekter. Ulike front end kan ha ulike måter å utveksle data. Dette gjør at man trenger muligheten til lett å kunne modifisere innhentingen av besvarelser og tidsforbruk. En utfordring med ulike front end, er å finne frem til måter å utveksle nødvendig data, slik at man i størst mulig grad tar høyde for nye utvekslingsformater som man kan støte på, og at man samtidig tar høyde for varierende dataformat.
Det er laget importeringsrutiner for å hente inn besvarelser og tidsforbruk fra SESE. Dataene som mottas, kommer som pakkede filer og som databasefiler, slik at implementasjonene er knyttet opp mot disse. Metodene er laget slik at man kan importere alle besvarelser eller kun begrensende mengder. Det kontrolleres også at en besvarelse ikke er importert fra før av. For å konvertere data mellom front end og JCAT, er det laget filer som definerer denne konverteringen. Disse filene inneholder tilstrekkelig data til at man kan hente ut nødvendig informasjon. Etter at man har eventuelt vasket dataene, kan innhentingsrutinen kjøres på nytt, og denne kan hente inn besvarelsene i JCAT, hvor de kan evalueres og skåres effektivt. Fordi importeringen sjekker om besvarelser eksisterer fra før av, effektiviseres importeringen av disse. Importering av besvarelser er ikke testet opp mot alle formater eller front end som eksisterer, slik at dersom nye front end må håndteres, kan man risikere å måtte modifisere importeringsmetodene noe. Innhentingen av besvarelser til å organisere disse i henhold til oppgaven de gjelder. Dette bør lett kunne modifiseres slik at man henter inn alle besvarelser og tidsforbruk. En annen mulig forbedring ligger i å benytte andre formater for å spesifisere konverteringen av besvarelser og tidsforbruk mellom front end og JCAT. I denne oppgaven gjøres innhenting av besvarelser kun via kommandolinje. En videreutvikling er å få et grafisk brukergrensesnitt på rutinene for å hente inn besvarelser fra front end til JCAT.

En av utfordringene med besvarelser som mottas via front end, er at man ikke kan garantere at subjektene leverer inn besvarelsene sine slik man angir i oppgaveteksten. Dette problemet har vist seg å inntreffe i flere tilfeller, slik at man behøver muligheten for å vaske dataene før de inngår i vurderinger av besvarelsene. Denne problemstillingen kan igjen medføre at man får feil i datagrunnlagets som benyttes for å vurdere prestasjonene. For å løse dette problemet, har jeg sett på hvordan man kan tillate datavasking av besvarelser før de tas inn i rammeverket. Måten det er gjort på i denne oppgaven, er ved å importere besvarelser og tidsforbruk data fra front end til rammeverk i to faser. Den første fasen henter besvarelsene inn på et område, hvor kan de gjennomgås manuelt. Den andre fasen henter så besvarelsene inn i rammeverket. Fordi vurderingene som gjøres, er offline, kan nettopp to-faset innhenting benyttes. Denne metoden kan ikke benyttes dersom vurderingen hadde vært online. Skal man håndtere samme problemstilling i online vurderinger, trengs andre tiltak. For å finne ut hvilke tiltak man kan sette inn her, trengs det flere studier.

7.2 Vurdering av besvarelser

7.2.1 Evalueringsmetoder

Metodene som skal evaluere besvarelsene automatisk, må for det første skilles godt nok fra skåring, slik at man lettere kan erstatte evalueringsmetodene med andre. For det andre må disse evalueringene utføres på en slik måte, at man ikke har avhengighet mellom testcase og den evaluerte besvarelsen. For det tredje, må evalueringer ikke være avhengig av strukturen på besvarelsen. For å se hvordan nye evalueringsmetoder kunne legges til, ble det gjort implementasjon av en enhetstestbasert evaluering, og en evaluering basert må menneskelige vurderinger av automatisk genererte differanserapporter. Fordi evalueringer tidligere ble påvirket av feil i besvarelser, måtte det finnes bedre måter å gjøre dette på. Gjennom å skille testcase og besvarelser med et grensesnitt som kunne hente inn besvarelsene under kjøring, kunne man unngå problemer som gjorde at man måtte restarte testene dersom besvarelser inneholdt feil. I tillegg til dette, er det laget ulike mekanismer for å håndtere transformering av opprinnelig struktur i besvarelser, til strukturer som er lettere å håndtere i forbindelse med vurdering av besvarelsene. Dette gjelder utpakking av besvarelser, men det finnes også en transformering i organiseringen av besvarelsene som innhentes fra SESE. Fordi man har fått denne fleksibiliteten, kan den utvides til også å hente inn nye evalueringsmetoder ved behov. Dette gjør rammeverket letter i stand til å generere nye skår automatisk. Evalueringsmetodene som benyttes i dag, kan ikke nødvendigvis benyttes på alle andre oppgavetyper. Dersom man trenger å legge til en ny oppgavetype, bør det derfor sjekkes om eksisterende evalueringsmetoder kan fungere, eller om en evalueringsmetode bør innføres. En annen ulempe, er at vurderingene av besvarelsene utføres i et delvis trygt miljø. Skal man hindre at besvarelsene kan lage problemer på maskinene de testes på, har man et par alternativer. Man kan enten distribuere besvarelsene, og utføre vurderingene på ulike maskiner. Dersom det oppstår feil på en maskin, vil det ikke påvirke de andre. Det andre alternativet er å kjøre evalueringene i et miljø som kan hindre at besvarelsene får tilgang til å endre kritiske deler av systemet de testes på. En ytterligere forbedring, når man skal legge til eller endrer evalueringene, kan være bruk av grafisk grensesnitt.

7.2.2 Skåring

Man vet hvordan enkeltbesvarelser skal skåres, men utfordringen var å se om de samme metoder kunne benyttes på delsvar. Dersom man i tillegg lar tidsforbruket være et kriterium for hvordan besvarelsen skal skåres, må tidsforbruket inngå i de vurderingene som gjøres. Når man har flere karaktersettere, må man også sammenligne resultatene, slik at man kan se hvilke forskjeller de ulike kriteriene påvirker det endelige resultatet. Løsningen på dette, var å innføre et regelsett som kunne

knytte sammen vurderinger av delsvarene, noe som ikke kunne gjøres tidligere. Dette regelsettet gjør at man kan utvide med nye kriterier som skal inngå i beregningen av skår, men samtidig tillater det å lage regler som kan tas i bruk på senere tidspunkt. Reglene som er implementert i denne oppgaven, er programkoder som må bygges sammen med resten av JCAT. Ved å gjøre evaluering og skåring av de enkelte delsvar først, kan man så benytte regelsettet til å bestemme den endelige skåren. Regelsettet tillater også at man evaluerer og skårer de enkelte delsvar i en rekkefølge, slik at dersom et subjekt feiler på en oppgave som de etterfølgende baserer seg på, utføres ikke de resterende vurderingene. I tillegg kan man benytte tankegangen rundt skåringsregler i forbindelse med fleksible evalueringer av besvarelsene. Fordi besvarelsene kan lastes inn i systemminnet, kan automatiske evalueringer foregå en del raskere enn tidligere. Dette gjør igjen at man kan kalkulere skårene raskere. Endringer av karaktersettere kunne også med fordel ha blitt gjort med grafisk grensesnitt. Man kan også vurdere å lage skript som endrer skåringsregler, og plasserer disse i egne tekstfiler.

7.3 Forberedelse til computer-adaptive tester

For at et rammeverk skal kunne sies å være klargjort for å computer-adaptive tester, vurderte jeg følgende til å måtte oppfylles. For det første må det kunne innhente og organisere besvarelser. Skal besvarelser gjelde deloppgaver, må også disse håndteres. Når man skal kalibrere oppgavene, trenger man sammenligningsgrunnlag for skåring, slik at man kan gjøre justeringer i forhold til vanskelighetsgrad. Man må kunne tilpasse evalueringene og skåringsmetodene i henhold til eksisterende oppgaver, og til oppgaver som kan komme på et senere tidspunkt. For at verdiene skal ha en viss troverdighet, må man i størst mulig grad fjerne kilder som kan forårsake feil i evalueringene og skåringene av besvarelsene. For at dette skulle kunne gjøres, ble besvarelser som kunne hentes fra SESE, innhentet og organisert. Besvarelsene ble tatt inn i rammeverket, for deretter å kunne evalueres og skåres på vanlig måte. Delsvar ble håndtert, delvis gjennom å gruppere dem sammen, og delvis gjennom å innføre regelsett for hvordan de enkelte oppgaver skal inngå i den totale vurderingen. Ved å tilby en mer dynamisk innlasting av klasser, kan man lettere endre evalueringskriteriene, i tillegg til at man kan vurderer besvarelsene uten å måtte initiere oppgavene med korrekt fungerende besvarelser, i den grad man måtte tidligere. På bakgrunn av dette kan vurderingene skje mer effektivt, og man får resultatene raskere enn tidligere. Ved å plassere skår i tabeller, kan man sammenligne resultatene fra ulike karaktersettere, og således bruke disse sammenlikningene til eventuelt å korrigere kriteriene ved behov. Det som ikke er gjort, men som det er lagt til rette for gjennom organisering av test/eksperiment og itemutvelger, er metodene som kalibrerer oppgavene med vanskelighetsgrad, og som plukker ut oppgaven. For at dette skal kunne gjennomføres, kan man trenge andre front end og/eller metoder for å sende oppgaver fra JCAT til front end.

7.4 Videre arbeide

På grunn av begrenset tid, og fordi det finnes flere mulige forbedringer og utvidelser av implementasjonene som er benyttet i denne oppgaven, foreslås det at følgende punkter kan inngå i videre arbeide:

- Man bør legge til nye metoder for vurdering av besvarelser, slik at man får mer omfattende skåring og skåringsregler enn i dag. Det er også flere oppgavetyper som kan legges til enn de som er tatt høyde for, slik som flervalgsoppgaver. Hvorvidt dette krever nye evalueringsmetoder, må i så fall undersøkes nærmere.

- Innhenting og organisering av besvarelser gjøres gjennom et verktøy med utgangspunkt i kommandolinje. Dette verktøyet kan med fordel gis grafisk brukergrensesnitt, og det samme gjelder også når man skal legge til eller endre evalueringer og skåringer.

- Bruken av Graybox for å skille testcase og besvarelser ble ikke ferdigstilt i denne oppgaven på grunn av manglende tid. Det finnes flere testcase, hvor denne teknikken med fordel kan benyttes. Dette arbeidet bør fullføres, slik at man får bedret vurderingene som gjøres av de ulike besvarelsene.

- Fordi man ønsker å gjøre tester av besvarelser så sikre som mulig, bør man i tillegg til uavhengighet mellom testcase og besvarelser, også se på hvordan disse kan kjøres i et sikkert miljø. Dette gjør at man i praksis er sårbar, og kan risikere at besvarelsene får tilgang til å endre deler av diskområder som de ikke bør få tilgang til. For å bedre testingen av besvarelsene, bør dette punktet undersøkes i fremtidig arbeid.

- Gjennom å distribuere besvarelsene i større grad, kan man oppnå økt effektivitet samt sikrere testing av besvarelser. Dersom man får kritiske feil på en datamaskin, kan andre maskiner lett overta. Derfor bør det undersøkes hvordan man kan få til slik distribuering.

- På grunn av at distribuering av besvarelser kan utføres på maskiner med ulike operativsystemer, bør JCAT også gjøres enda mindre avhenging av operativsystemet enn det er i dag. Dette vil også gjøre JCAT lettere å overføre mellom ulike operativsystemer.

- Dersom man skal få bedre kontroll av potensielle feil i datagrunnlaget ved computer-adaptive tester, kan to-faset importering være en tidsmessig flaskehals. Det kan derfor være interessant å se på andre, alternative løsninger.

Fordi man med disse metodene forbereder JCAT for computer-adaptive testing, er det fremdeles noen gjenstående utfordringer knyttet til sammenligning og presentasjon av de ulike karakterene som bør adresseres. Blant annet kan det være nødvendig å se om metodene som presenterer karakterer kan utvides, slik at man får mer informasjon om de andre karaktersetteres vurderinger. Alt burde dermed ligge til rette for at man kan utvikle nødvendige metoder for å kalibrere oppgaver, og å plukke ut oppgaver som skal presenteres for subjektene. Disse må så ha metoder for å kunne sende oppgaver ut til front end, basert på de vurderinger som gjøres i rammeverket.

Ordliste

Allokering: Fordele plassering.

Backlog: Liste over arbeid som venter på å bli gjort.

Computer-adaptiv test: En test, eksperiment hvor rekkefølgen på oppgaver bestemmes gjennom dataassistert vurdering av besvarelser.

Delsvar: En besvarelse som inngår som del av en større besvarelse.

Deloppgave: En oppgave som er en del av en større oppgave.

Eksperiment: Som test, men oppgavene kan komme i ulike rekkefølger. Eksperiment benyttes for å teste hypoteser.

Evaluering: Verdinøytral måling av kvaliteter ved besvarelser.

Gråboks (Graybox): En svartbokstesting av besvarelser interne struktur.

Intern struktur: Organisering av klasser og metoder, eller av andre komponenter som inngår i besvarelsen.

Item: En oppgave med tilhørende beskrivelse og innhold.

Itemutvelger: Også kalt ”Item selector”, og en algoritme for å velge ut oppgaver som skal gis til subjektene.

Kalibrering: Fastlegging av skala på et måleinstrument og angivelse av fysisk størrelse.

Skåring: En karaktersetting eller rangering av besvarelse.

Subjekt: En person som tar en test.

Svartbokstesting: En testing av besvarelser uavhengig av besvarelsenes interne struktur.

Test: Eksperiment, prøve, metode i psykologien karakterisert ved en standardsituasjon som forskjellige individer kan stilles overfor, og hvor man er interessert i de individuelle forskjeller i prestasjoner og reaksjoner. I tester kommer oppgavene i en bestemt rekkefølge.

Verdinøytralitet: Man tar ikke stilling til hvor verdifulle ulike egenskaper er.

Verdisystem: Et system for hvordan man skal vektlegge hvor verdifulle ulike egenskaper er.

Vurdering: Evaluering og skåring av besvarelse.

Kildeliste

Ala-Mutka, K. M. (2005). "A Survey of Automated Assessment Approaches for Programming Assignments." Computer Science Education 15: 83-102.

Arisholm, E. og P. M. Chen (2004). "An automated feedback system for computer organization projects." Education, IEEE transaction on 47(2): 232-240.

Arisholm, E., D. I. K. Sjøberg, et al. (2002). "A web-based support environment for software engineering experiments." Nordic J. of Computing 9(3): 231-247.

Bond, Linda A (1996), ”Norm- and Criterion-Referenced Testing”, ERIC ID: ED410316, fra http://www.ericdigests.org/1998-1/norm.htm, 6. juni 2008

Cheang, B., A. Kurnia, et al. (2003). "On automated grading of programming assignments in an academic institution." Computers & Education 41(2): 121-131.

De Gruijter, D. N. M. (1986). "The Use of Item Statistics in the Calibration of an Item Bank." Applied Psychological Measurement 10(3): 231-237.

Edwards, S. H. (2003). Using test-driven development in the classroom: Providing students with automatic, concrete feedback on performance. EISTA'03.

Marciniak, John J. (2002). Encyclopedia of Software Engineering, Second Edition, Wiley & Sons Inc, New York, 2002. ISBN: 0-471-37737-6.

Highsmith, J, Cockburn, A. (2001). "Agile Software Development: The Business of Innovation." Computer 34(9): 120-122.

Luecht, R., T. Brumfield, et al. (2006). "Research Articles: A Testlet Assembly Design for Adaptive Multistage Tests." Applied Measurement in Education 19(3): 189 - 202.

Mike Joy, Nathan Griffiths, et al. (2005). "The BOSS Online and Assessment System." ACM Journal on Educational Resources in Computing 5(3): Article 2.

Morris, D. S. (2003). Automatic grading of student's programming assignments: an interactive process and suite of programs Frontiers in Education, 2003. FIE 2003. 33rd Annual.

Nghi Troung, P. R., Peter Bancroft (2004). Static Analysis of Students' Java Programs. Sixth Australian Computing Education Conference (ACE2004), Dunedin, New Zealand, Australian Computer Society.

Ronald K. Hambleton, R. W. Jones (1993). "Comparison of Classical Test Theory and Item Response Theory and Their Applications to Test Development." Instructional Topics in Educational Measurement.

HISP. (2007). "Health Information Systems Programme." from http://hips.ifi.uio.no:8080/display/HISP/Health+Information+Systems+Programme+-+HISP.

tigris.org. (2008). "Tortoise SVN." from http://tortoisesvn.tigris.org/.

Wikipedia.org. (2008). "Computer-adaptive test." fra http://en.wikipedia.org/wiki/Computer-adaptive_test#cite_note-WeissKingsbury-0.

Wikipedia.org. (2008). "Criterion-referenced test." Fra http://en.wikipedia.org/wiki/Criterionreferenced_test.

Wikipedia.org. (2008). "Eksperiment." fra http://en.wikipedia.org/wiki/Experiment.

Wikipedia.org. (2008). "Evaluering." fra http://en.wikipedia.org/wiki/Evaluation.

Wikipedia.org. (2008). "Norm-referenced test." fra http://en.wikipedia.org/wiki/Norm-referenced_test.

Wikipedia.org. (2008). "Regression analysis." fra http://en.wikipedia.org/wiki/Regression_analysis.

Wikipedia.org. (2008). "Scrum (Development)." 21. mars fra http://en.wikipedia.org/wiki/Scrum_(development).

Wikipedia.org. (2008). "SMART", 21. mars 2008 fra http://en.wikipedia.org/wiki/SMART_(project_management).,

Wikipedia.org. (2008). "Pivot table." Hentet 26. mai, 2008, fra http://en.wikipedia.org/wiki/Pivot_table.

Vedlegg

Vedlegg A – Delsvarshåndteringer

For å hente ut et bestem delsvar, ble de identifisert gjennom tall eller bokstav. Denne koden finner frem til korrekt delsvar, basert på denne identifiseringen. For å kunne evaluere eller skåre delsvar, trenger man å kunne finne frem til det enkelte delsvaret som evalueringen eller skåringen gjelder, og det er dette denne koden gjør.

```
/**
 * Answers where the next answer is based on previous answer.
 * TODO: rename to SubTaskAnswer

 */
public class SubTaskAnswer extends Answer{
 …
public Answer getPart(int part) {

  for (Answer answer: subTask) {
    if (answer instanceof FileAnswer) {
     FileAnswer fa = (FileAnswer)answer
     Pattern pattern = Pattern.compile("^(.*)(\\d+|\\w+).zip",
Pattern.DOTALL);
     Matcher match = pattern.matcher(fa.getTargetFileName());
     if (match.find()) {
      if (!match.group(1).startsWith("task") &&
!match.group(1).startsWith("item") ) {
       System.err.println("Found match " + match.group(2));
       try {
        int partcomp = Integer.parseInt(match.group(2));
        if (partcomp == part)
         return answer;
       } catch (NumberFormatException ne) {
        int partcomp = (match.group(2).toLowerCase().charAt(0) - 'a')+1;

        if (partcomp == part)
         return answer;
       }
      }
     }
    }

  }
  return subTask[0];
 }//get part
```

Vedlegg B – Innhenting av manuelt satte karakterer/skår

Karakterer eller skår leses inn fra en fil, plassert på samme område som besvarelsen. Denne filen identifiserer karaktersetteren og besvarelsen den gjelder, og koden nedenfor viser hvordan data fra denne kan hentes inn.

```
/**
  * Reads score from file
  */
 public int getIntScore(Answer answer) throws JCATException {
  int score = -1;
  String fileName = GLOBAL.ITEM_ANSWER_PATH + answer.getItemName() +"\\"+
answer.getSubjectName() + "\\" + defaultFileName + "-" + getRaterID() +
((FileAnswer)answer).getTargetFileName().replace(".zip", "") +
defaultExtension;
  File file = new File(fileName);
  if (file != null && file.exists()) {
   String [] fileContents  = new
FileFunctionality().readFromFile(file.getAbsolutePath());
   if (fileContents != null) {
    //read score
    Pattern pattern = Pattern.compile("^\\s*(\\w+),(\\w?),(\\d+)");
    Matcher matcher = null;
    for (String line : fileContents) {
     if ((matcher = pattern.matcher(line)).find()) {
      score = Integer.parseInt(matcher.group(3));
     }
    }
   }
  }
  return score;
 }
```

Koden er knyttet til menneskelige evalueringer av besvarelser, og vil kun utføres i den forbindelse.

Vedlegg C – Lisenser

I dette vedlegget vises et par eksempler på lisenser inne åpen kildekode.

Lisens for bruk av SCRUM figur:

*License for Permission is granted to copy, distribute and/or modify this document under the terms of the **GNU Free Documentation license**, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation license".*

GNU General Public License

Version 2, June 1991

```
Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place - Suite 330, Boston, MA  02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
```

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder

saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the

executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and an idea of what it does.
Copyright (C) yyyy  name of author

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
type `show w'.  This is free software, and you are welcome
to redistribute it under certain conditions; type `show c'
for details.
```

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright
interest in the program `Gnomovision'
(which makes passes at compilers) written
by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.