Forord

Prosjektet *BIRI 2* ble påbegynt januar 2000 av Jarle Rudihagen, Robert Strand og Frank Svanheld. Prosjektet ble til etter et forslag fra oppdragsgiver Erik Hjelmås, for å møte kravene til hovedprosjekt for høgskole ingeniør i datateknikk ved Høgskolen i Gjøvik. Da gruppen, som kjente hverandre fra før da vi hadde jobbet sammen i prosjektgruppe tidligere i utdanningen, ble presentert for dette forslaget ble det vel mottatt av oss. Det virket som et interessant prosjekt, og samtidig fikk vi anledning til å bli kjent med det alternative operativ systemet *Linux*. Operativsystemet var delvis brukt av prosjekt medlemmene fra før, men nå søkte vi mulighetene til å utvikle programvare for denne plattformen.

Erik Hjelmås hadde drevet med det helhetlige *BIRI*-prosjektet over lengre tid og driver i disse dager med en doktor avhandling rundt det temaet BIRI prosjektet berører(`www.hig.no/~erikh`).

Ved Høgskolen i Gjøvik takker vi:
Erik Hjelmås for å være veileder, oppdragsgiver og for hjelp innen gjenkjenning og bildebehandling. **Ivar Farup**, for hjelp i *C++* og *Qt* programmering. **Tom Røise**, for tidligere prosjekt veiledning som har vært nyttig. **Frode Haug**, for opplæring i *C++*. Og **Øyvind Kolloen** for MySQL assistanse.
Utenfor Høgskolen i Gjøvik takker vi:
Qt interest group ved **TrollTech**, for behjelpelighet når det gjelder *Qt* programmering. **MySQL++ mailing list** og spesielt **Sinisa Milivojevic**, for hjelp i bruken av *MySQL++* programmerings *api*.

Innhold

1 Innledning

1.1 Organisering av selve rapporten

Kap.1 Innledning

Kap.2 Kravspesifikasjon

Kap.3 Gjenkjennings algoritmer

Kap.4 Analyse

Kap.5 Design

Kap.6 Implementasjon

Kap.7 Testing

Kap.8 Installasjon og bruk

Kap.9 Diskusjon

Kap.10 Underskrift

Kap.11 Litteraturliste

Innledningen er ment å gi leseren av rapporten et innblikk i hva prosjektet dreier seg om, og si litt om oppgavens avgrensning. Samt si litt ombruksområdene til systemet. Den sier også en god del om hvilke arbeidsmetode/utviklingsmodell vi har lagt oss på. Det spesifikke kravene til systemet er beskrevet i kapittel 2, det er disse kravene som er lagt til grunn for selve utviklingen og som i størst mulig grad skal være dekket i den endelige løsningen. Det komplette kravspesifikasjons dokumentet kommer med som vedlegg B, men hovedtrekkene er tatt med i kapittel 2.

Med andre ord kan man si at kapitlene:3, 4, 5 , 6 , 7 og 8 er sterkt knyttet til det som defineres i kapittel 2. Kapitlene som tar for seg analyse og design, konsentrerer seg i hovedsak om metodikk. Vi har laget et komplett analyse og design dokument der vi benytter *UML* rammeverket. Dette dokumentet følger med som vedlegg C.

Analysen gir det bilde over hva systemets skal gjøre og definerer funksjonelle og operasjonelle krav. Design sier noe om hvordan systemet skal designes for å kunne utfylle krav som stilles til systemet. Kapitlet om implementasjonen forteller om hvordan selve kodingen av systemet ble utført, samt prosesser rundt dette eks. standarder for koding, verktøy, versjonshåndtering og dokumentering. Testing og kvalitetssikrings kapitlet tar for seg hvordan kvalitetssikring ble ivaretatt gjennom utviklingsforløpet, og hvlike tester og metoder for testing som er blitt brukt. Det hele oppsummeres i en konklusjon der det går frem hvilke resultat som ble oppnådd i forhold til hva vi hadde som målsetting, hvordan utviklings prosjektet har forløpt, hva kunne ha vært gjort annerledes, og lærdom/erfaringer prosjektet har gitt oss.

1.2 Definisjoner

Ord/uttrykk Definisjoner

Algoritme Oppskrift for hvordan et gitt problem løses

api *Application Programming Interface*

Applikasjon Program for en datamaskin.

Biometrisk Bruker fysiologiske og atferdsmessige egenskaper til et menneske og ved hjelp av disse gjenkjenne f.eks. et menneske

BIRI1 En del av det helhetlige BIRI prosjektet. Skal i dette prosjektet detektere et ansikt foran et web kamera.

BIRI3 Del av BIRI prosjektet. Skal animere et ansikt og lage en talegenerator for bruke i kommunikasjon med bruker.

browse Se på. Se over/gjennom. Browser mye omtalt som nettleser. Leser av HTML dokumenter.

C Et høynivå programmeringsspråk utviklet ved Bell Labs som kan manipulere maskinen med et lavnivå språk.

C++ Et objekt orientert versjon av programmeringsspråket C laget av Bjarne Stroustrup. Har blitt populært fordi det kombinerer tradisjonell C programmering med mulighetene i Objekt Orientert Programmering (OOP).

check-box Avkryssningsboks.

Debugger Program som hjelper utviklere å renske program for feil. Hovedfunksjonen er at man skal kjøre programmet steg-for-steg og sjekke hva som skjer underveis i programmet.

eigenface Et egenskaps ansikt der man har trukket ut spesielle egenskaper i ansiktet hos ulike personer. Brukers i i metoder der man bruker at ansikter er et vektet kombinasjon av disse eigenfaces.

Egenskapsvektor En vektor med tall som beskriver egenskapene til et bilde. Man kan trekke ut visse egenskaper og samle disse i en vektor.

Euclidske Avstand mellom punkter i et vektor rom i formen $d_{bE} = \sum_{ij}(I_{ij} - T_{ij})^2$

FileShare1 Egen utviklet applikasjon for fildeling via internett

Gabor En effektiv strategi for å dra både *koherente* and inkoherente teksturell informasjon fra bilder, slik som detaljerte teksturer, er bruken av 2-D Gabor fase koeffisienter.

Gannt Diagram som viser tids og ressurs bruken fremover i en tidsperiode.

Gdb Er en kildekode avluser, laget av Free Software Foundation i USA

GNOME GNU Object Model Environment - Et GUI-basert brukermiljø for Linux utviklet som en del av GNU prosjektet. Det er et alternativ til KDE. Red Hat Software (www.redhat.com), en ledende Linux distributør støtter utviklingen av GNOME.

GNU Gnu's Not Unix - Et prosjekt sponset av Free Software Foundation som utvikler et komplett program miljø inkludert operativsystem kjerne og verktøy, editor, kompilator og debugger.

GUI Graphical User Interface - Et grafikk-basert brukergrensnitt som inneholder ikoner, dra-ned menyer og styres med mus. GUI har blitt standarden for hvordan brukere benytter datamaskiner.

HTML Hyper Text Markup Language - dokumentformat brukt på WWW. Web sidene er bygd opp av HTML-tags eller kodet sammen med teksten. HTML definerer utseende på siden, skrifttyper og grafiske elementer samt hypertekstlenker til andre dokumenter på web. Hver link inneholder adressen til en webside på samme serveren eller en annen server lokalisert hvor en i verden.

IEEE-730 The Institute of Electrical and Electronics Engineers - Standard for kvalitetssikring.

Java Objekt Orientert programmerings språk utviklet av Sun microsystems (`www.sun.com`).

KDE K Desktop Environment - Et GUI basert bruker grensesnitt for UNIX arbeidsstasjoner. Det er et komplett skrive bords miljø, omtrent som Windows og Mac, med sine egne unike stil og funksjonalitet. Programkoden er distribuert fritt og vedlikeholdes av utviklere verden over. Utbredt brukt under Linux.

Koherent Samvarierende. Samsvingende.

Korelasjon Det er et mål for systematisk samvariasjoner mellom to stokastiske variable X og Y. I vårt tilfelle bilder.

Kompilere Å oversette et program skrevet i høynivå språk til maskinspråk.

Latex Et avansert program for dokumentasjons-produksjon, altså typesetting av artikler, bøker, brev og annet. Det som skiller latex fra andre programmer brukt i dokument produksjon er at det ikke er interaktivt, men ligner mer på programmering.

Linux En versjon av UNIX som kjører på x86, Alpha og PowerPC maskiner. Linux er et open source program, som dermed er fritt tilgjengelig, men en full distribusjon av Linux sammen med teknisk support og hjelp er tilgjengelig for en liten avgift fra distribuører som Red Hat Software (www.redhat.com) og Caldera (www.caldera.com). CD'ene som blir distribuert, inneholder komplett programkode sammen med hundrevis av verktøy, tillegg og kjekke programmer.

Makefile Script for kompilering og linking av kildekode i et prosjekt.

MYSQL Et database manager system som er utbredt blant annet med stor bruk hos NASA. Gratis tilgjengelig for utvikling. Lisensiert ved salg av applikasjoner. Regnet som en av de kjappeste database motorene i dagens marked.

MYSQL++ Et programmerings bibliotek for C++. Omskrevet C bibliotek. Vedlikeholdes i dag med ansvar hos Sinisa Milivojevic (sinisa@mysql.com).

NevraleNettverk Hoved karakteristikken for nevrale nettverk er at de har evnen til å lære komplekse ikke-lineære input-output forhold, bruke sekevensielle lærings rutiner, og å tilegne seg dataene.

ObjektOrientert Anser alle ting som objekter i en verden. Software beskrivelser av tenkte ting i virkeligheten.

OOA/OOD Objekt Orientert Analyse/Design -Undersøkelse av et problem ved å modellere et som en gruppe med sammenhenger mellom objekter.

OpenSource Gratis programkode til et program, som blir gjort tilgjengelig for utvikler-milepæler generelt. Bakgrunnen er tanken om at en større gruppe programmerere som jobber med ett program til slutt vil gi et mer brukbart og feilfritt produkt til alle, spesielt fordi mennesker har sett gjennom programkoden. Gjennomsyn av koden er kjent som en av de viktigste måten å hindre programkode med masse feil i, men blir ofte ikke vektlagt nok, hvis i det hele tatt, av program selskapene. Gjennomsyn av kode e en naturlig følge av open source.

Qt Qt er et objekt orientert programmeringsbibliotek utviklet av det norske firmaet Troll Tech[] og skrevet i C++ . Qt forenklet GUI programmering, og er tilgjengelig for flere plattformer (Unix/Linux, Windows, Machintosh). Dette medfører at program basert på Qt kan kompileres og kjøres på alle disse plattformene.

real-time Sanntid. At man skal ikke oppleve noen forsinkelse mellom input data og output data.

rtf RichTextFormat. Tekst med formatering.

SDI Single Document Interface - Arkitektur for layout

strace Programmet strace [programfil] lister opp alle kall til kjernen som gjøres av programmet mens det kjører. På denne måten kan det være mulig å oppdage rare ting i programmet.

tmake Verktøy for å lage kompilering og linke script, *Makefile,* for C++.

Trigge Starte en prosess ved den tidspunktet da denne prosessen skal settes i gang.

UML Unified Modeling Language - modellerings språk for Objekt Orientert Systemutvikling

UNIX Et flerbruker, fleroppgave (multitasking) operativ system som er vidt utbredt som operativsystem for arbeidsstasjoner og spesielt servere.

USB Universal Serial Bus. En ny og meget aktuell teknologi for å tilknytte eksterne medium til en datamaskin. Kan knytte mange enheter til en bus da man kan teoretisk kan henge på 127 enheter på en bus.

1.3 Problemområde og avgrensning

Oppgaven vår er en del av et større prosjekt *BIRI(Biometric Inteligent computeR Interface).* Det helhetlige prosjekt har til målsetting å utvikle et intelligent interaktivt multimedia system hvor brukeren skal kunne kommunisere med en datamaskin i størst mulig grad vha. lyd og bilde i stedet for den tradisjonelle måten med mus og tastatur. En annen viktig forutsettning for prosjektet er at det skal utvikles som et *OpenSource* produkt, og at det skal være mest mulig plattform uavhengig. Men vi skulle spesielt utvikle løsningen for en *Linux* plattform.

Vi i Biri2 skal basere oss på å få et input bilde fra et kamera. Dette input bilde skal inneholde et ansikt, *BIRI1* har ansvaret for å ta seg av kamera delen for å detektere et ansiktet. Ut fra input bilde skal vi gjøre en spørring mot en database som inneholder ansikts bilder, *egenskapsvektor* og tilhørende data som for eksempelvis fornavn, etternavn og en liten kommentar. Systemet skal videre ved hjelp av *algoritmer* og matching mot bilder i databasen regne seg frem til hvem personen på input bilde er. Når vi i Biri2 har klart å gjenkjenne personen på input bilde skal det *trigges* en prosess som starter en animasjon bestående av et ansikt som forteller brukeren av systemet hvem personen på input bilde mest trolig er vha talegenerering. Animasjons og tale genererings delen er det *BIRI3* som har ansvar for.
Vi i BIRI2 har altså som oppgave å implementere:

* Database delen i systemet, dvs. sette opp selve databasen

* Layout og funksjonalitet for innlegging av bilder, *egenskapsvektor* og tilhørende data.

* Layout og funsjonalitet for vedlikehold av databasen, dvs. søk etter personer på fornavn og etternavn, oppdatering av data som er lagret og sletting av data.

* Layout, funksjonaltiet og algoritmer for gjenkjenning av person på input bilde til systemet. Samt mulighet for valg av algoritme som skal brukes i gjenkjenning prosessen.

* Tilrettelegge slik at *applikasjon* kan kjøres uten kamera, dvs at input bilde lastes opp fra fil.

* Tilrettelegge slik at *applikasjonen* kan kjøres uten animasjon og talegenerering, dette innebærer å lage en layout del som presenterer resultatet av en gjennomført gjenkjenning.
  I utviklingen av vår løsning har vi avgrenset oss til å bruke kjente algoritmer for ansikts gjenkjenning. Vi har med andre ord ikke utviklet noen nye revolusjonerende algoritmer da det ville vært for omfattende. Men vi har i vår løsning lagt til rette for at nye/andre algoritmer skal kunne implementeres i systemet på en enkel måte.

1.4 Målgruppen

Målgruppen for systemet vi utvikler er i hovedsak brukere av *Linux* og *OpenSource* programmer, men løsningen er implementert mhp. plattform-uavhengighet slik at flest mulig skal kunne gjøre seg nytte av den. Vi ønsker å utvikle en *applikasjon* som lett kan benyttes av oppdragsgiver for å teste ut algoritmer for gjenkjenning av mennesker ved hjelp av ansiktet, samt å være et system for identifikasjon og en database for registrering av mennesker og deres ansikts bilde.

Selve rapporten er i første rekke ment for de som skal evaluere vårt prosjektarbeid. Den vil selvfølgelig også være for generelt data interesserte og ikke minst studenter , særlig med tanke på videreutvikling av systemet. Rapporten er også tiltenkt som en veiledning for framtidige brukere av systemet, som vår visjon for hva systemet skal gjøre og hvordan dette skal gjøres.

1.5 Formålet

Vi i gruppen har stor interesse for *Linux* og *OpenSource* programmer, vi finner det inspirerende å se at det finnes gode alternativer til f.eks. *MS Windows* som etterhvert kan bli slagkraftige konkurrenter og fullt på høyde med lisensiert programvare. Når vi da fikk muligheten til å delta i utviklingen av et system som det oss bekjent ikke fantes noe god *OpenSource* løsning på, grep vi sjansen. Vi får gjennom dette prosjektet utvidet vår horisont i *Linux* verden ved at vi tar i bruk for oss nye verktøy og at vi må tilegne oss ny kunnskap om disse. Samtidig får vi gjort oss nytte av tidligere kunnskap vi har tilegnet oss ved studiet ved høgskolen og utvidet denne ytterligere. Tenker da særlig på programmering, systemutvikling og *GUI* design. Samtidig tilegner vi oss verdifull erfaring om jobbing i prosjekter mhp. planlegging, delegering, koordinering og styring.

1.6 Prosjektgruppens bakgrunn og kompetanse

* Jarle Rudihagen: Allmenfaglig studieretning med EDB
* Frank Svanheld: yrkes skole elektronikk, forkurs ingenørhøyskole
* Robert Strand: yrkes skole elektronikk/m fagbrev, forkurs ingenørhøyskole

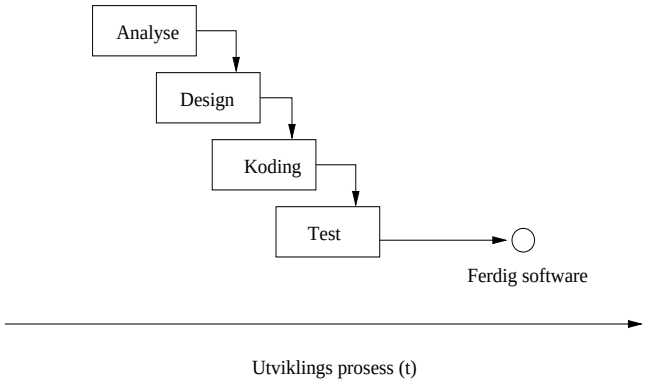
Alle tre gruppe medlemmene er i ferd med å avslutte studiet ved ingeniørhøyskolen datateknisk linje.

1.7 Arbeidsformer

Vi valgte en litt spesiell måte å håndtere leder rollen i gruppen. Det vil si vi har latt gruppe-leder vervet gått på omgang slik at alle har fått prøve seg. Dette fordi vi mener det er viktig at alle medlemmene har behov for å få med seg den verdifulle erfaring dette gir, selv i et lite prosjekt med få deltakere. Lederen har hatt det overordnede ansvar for å følge opp og sørge for at ting blir gjort. Selve oppgave fordelingen har vært fordelt mest mulig etter den enkeltes interesser/ønsker. Utviklingen har i all hovedsak foregått på samme rom(kontor) på skolen, eller at hvert prosjektmedlem har fått tildelt en oppgave som skulle analyseres og løses. Mye av koordinering gått nærmest automatisk i og med at vi har jobbet så tett sammen, men vi har hatt situasjoner der vi måtte delegere og koordinere aktivitetene.

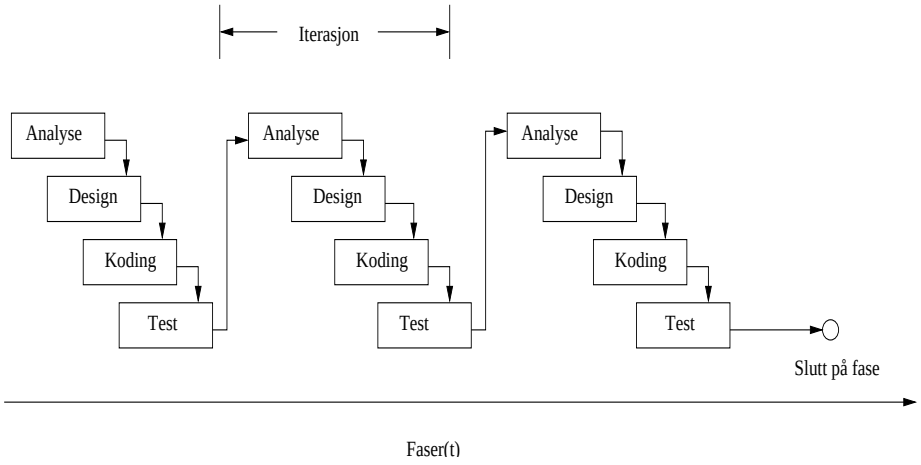
I systemutviklings litteratur er det i hovedsak snakk om to måter å drive *systemutvikling* på: **strukturerte metoder** og **objekt orienterte metoder**. I objekt orienterte programmerings verktøy er det objekt orienterte metoder som råder, og det er det vi har utviklet vårt system etter. Det er opp gjennom tidene utviklet flere typer utviklings modeller for å lette arbeidet med systemutvikling, de mest kjente er: **fossefallsmodellen**, **evlusjonærmodellen**, **spiralmodellen**, **gjenbruksbasert utvikling modell** og **inkrementellmodell**. Vi skal ikke her gå inn på å beskrive de ulike modellene, deres fordeler og ulemper men synes det er riktig å ha nevnt dem. Vi har nemlig basert vårt systemutviklings prosjekt etter den nyeste tilveksten innen systemutviklingsmodeller *RUP* (Rational Unified Process). Det spesielle med denne prosessen er at den bygger på og forsøker og ta opp i seg de beste elementene fra de best kjente modellene som nevnt ovenfor. Det er ikke dermed sagt at RUP løser alle problemer og vanskeligheter som ligger i et sytemutviklings prosjekt, men den er et spennende og godt alternativ til de mer tradisjonelle velkjente modellene.

Av figur 1.1 ser man den tradisjonelle fossefallsmodellen, i et utviklingsforløp vil denne den modellen gå gjennom de aktivitetene som fremgår av figuren. Fossefallsmodellen tar i veldig liten grad høyde for iterasjoner, det vil si at når man har avsluttet en aktivitet går man ikke tilbake og gjør endringer i særlig grad.



figur 1.1 viser fossefallsmodellen

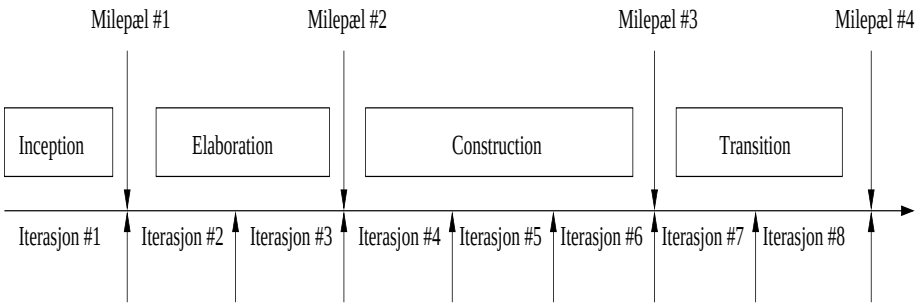
I motsetning ser vi av figur 1.2 hvordan *RUP* legger opp til flere iterasjoner, og hvor hver av disse inneholder en form for fossefallsmodell med dens aktiviteter. *RUP legger* opp til et mer dynamisk utviklingsforløp, som i høy grad støtter endringer underveis i utviklingsforløpet. Videre støtter *RUP* opp under inkrementell utvikling, hvor man gjennom hele prosjektforløpet har fortløpende risikovurderinger og gradvis implementerer funksjonalitet.



figur 1.2 Iterasjoner i RUP

Figur 1.3 gir en oversikt over RUP sin fase inndeling, videre følger et forslag til hva som bør inngå i de ulike fasene. Fasene avsluttes med milepæler, nødvendige tilpasninger og definering av milepæler må gjøres for å i møte komme

de enkelte utviklingprosjekters behov.



figur 1.3 RUP sin fase inndeling

RUP sin inndeling i faser:

1.Inception

* Formulere mål
* Oppnå enighet om mål
* Identifisere fundamentale *Use Case*
* Estimere innsats/tids/ressursforbruk
* Utarbeide faseplan med tilhørende milepæler

Artifakter fra inception fasen:

* Visjonsdokument som sier noe om det endelige målet
* Faseplan/fremdriftsplan (milepæler)
* Kostnadsplan
* Detalj-plan for neste fase

2. Elaboration

* Krav spesifisering
* *Use Case* detaljering
* Vurdere system arkitektur
* Risiko vurderinger

Artifakter fra elaboration fasen:

* Kravspesifikasjons dokument
* Funnet frem system arkitekturen
* Identifisert og avklart risikomomenter

3. Construction

* Implementasjon av systemets forskjellige komponenter, parallelt der hvor det ikke finnes sekvensiell avhengighet mellom komponentene.
* Drive nær oppfølging av implementasjons prosessen, stor vekt på kvalitetssikring, testing og dokumentasjon.

Artifakter fra construction fasen:

* *Betaversjon* av systemet som tilfredsstiller kravspesifikasjonen

4. Transition

* Feilsøking/retting på systemet
* Etablere driftsmiljø, installasjon
* Opplæring av brukere
* Konverter/integrere mot gamle systemer.

Artifakter fra transition fasen:

* Ferdig system som tilfredsstiller kundens behov

* Opplæring av brukere gjennomført

* God dokumentasjon skal foreligge

Karakteristika ved RUP:

* Den legger opp til en eller flere iterasjoner innenfor en fase se fig som viser fossefall inne i iterasjon
* Use Case drevet utvikling
* Fokus på arkitektur
* Gradvis utvikling vha. inkrementer
* Ivareta planleggbarhet og styring

For å sammenfatte litt om *RUP*: Modellen legger opp til flere iterasjoner slik at eventuelle endringer og uforutsette risikomomenter lettere skal kunne fanges opp. Modellen er *Use Case* drevet slik at man angriper problemet mest mulig grad slik det vil fortone seg i virkeligheten. Man starter med de *Use Case* som er mest essensielle og komplekse slik at man så tidlig som mulig i utviklings prosessen for avklart store risikomomenter. Modellen legger opp til at det utvikles inkrementer, slik at man gradvis implementerer funksjonalitet, på en slik måte at man bryter ned oppgaven i mindre komplekse deler for å unngå å miste oversikt og motivasjonen. Dette innebære også at man kan teste gjennom hele prosessen, samt lettere holde rede på hvor man befinner seg i forhold til fremdriftsplan. Styring ,koordinering og planleggbarhet blir dermed i varetatt.

Problemer i utviklingsprosjekter å er å vurdere risiko momenter som kan dukke opp lenger ut i utviklings prosessen. Dette er spesielt vanskelig å estimere i utviklingen av avanserte datasystemer. Det finnes mange store prosjekter som har vært mislykket, fordi risikoen som ligger i prosjektet har blitt oversett eller undervurdert. Derfor er det iallefall viktig å ha en modell som tar tak og vurderer risiko momentene tidlig og under hele prosessen. Vi mener *RUP*, som det "nye" paradigmet, har trukket mye av det beste av dette ut i hvert enkelt modell. Den modellen av de tradisjonelle modellene som tar risiko vurderingen mest for seg er **spiralmodellen**, der risiko-vurdering er en egen fase i en utviklings sirkel (1/4 av sirkelen).

1.8 Terminologi

Vi benytter oss av en del ord og uttrykk som er engelsk eller av en slik karakter at de trenger litt nærmere forklaring. Bruken av disse ordene er i stor grad "uunngåelig" og fordi vi mener at en del engelske-relaterte uttrykk i godt kan forklares på norsk uten at det kan virke litt kunstig. Forskjellige definisjoner har vi samlet i kapittel 1.2 under *definisjoner*. Men vi har så stor grad som mulig prøvd å for-norske terminologien uansett om de fleste fagskrifter innenfor området vi driver er engelsk språklige.

1.9 Praktiske opplysninger

I denne rapporten bruker vi forskjellige stiler:

* *kursiv* brukers når man bruker uttrykk som blir nærmere forklart og definert i terminologi-listen i kap. 1.2.

* **bold** skrift brukes når man vil understreke og innføre en ny forklaring/definisjon.

Merk også at:

* Referanser finner du som et eget kapittel bak i rapporten (kap. **??**). Litteratur i bokform er er på formen [1] hvor tallet er referanse nummeret.

* Denne rapporten er skrevet i LyX(`http://www.lyx.org`), en *dokument prosessor* basert på LaTeX. Grafikk og diagrammer er tegnet i XFig og *UML* er tegnet i TeleLogic UMLsuit(`http://www.telelogic.com`). Ellers har vi brukt GIMP (`http://www.gimp.org`) for all bilde behandling og bearbeiding. Kildekoden er prosessert og dokumentert vha Doxygen (`www.doxygen.org`).

* Hele rapporten, dokumentasjon og kilde kode skal legges ut på prosjektets hjemmeside http://prosjekter.hig.no/biri2 `http://prosjekter.hig.no/biri2`. All koordinering av denne siden blir overlatt til Høgskolen i Gjøvik ved prosjektets innlevering.

2 Kravspesifikasjon

Det komplette krav spesifikasjons dokumentet følger som et vedlegg B til selve rapporten pga sin størrelse. Trekker her ut de viktigste aspektene rundt den.

2.1 Bakgrunn

Utvikling av software for *Linux* er i stor grad avhengig av frivillig innsats. Dette fører til at det for enkelte typer software tar det lang tid før den er tilgjengelig, men det sikrer samtidig at den tilgjengelige softwaren blir gjennomgående testet. *Linux* har i den senere tid utviklet seg fra og være et operativ system kun for serverapplikasjoner til å bli et system med kraftig *GUI* (blant annet *GNOME* og *KDE*) og gode bruksegenskaper også for arbeidsstasjoner. Siden dette er et relativt nytt fenomen er det fortsatt stort behov for utvikling av standard programvare. Om vi vil ha flere til å bruke *Linux* er det essensielt at programvare for å gjøre vanlige operasjoner tilgjengelige. Det har til nå ikke vært så lett tilgjengelige noe godt *Open Source* alternativ til en interaktiv kommunikasjon med en datamaskin (vi vet bl.a. at Microsoft satser tungt på området for tiden).

BIRI 2 består i å gjenkjenne ansikter lagret som bilder i en database, database operasjoner, samt å lage en god layout for bruker interaksjon med applikasjonen.
Biometriske systemer bruker fysiologiske og atferdsmessige egenskaper til et menneske og ved hjelp av disse gjenkjenne f.eks. et menneske.
Biometriske systemer kan brukes til mange ting bl.a.:

- banktjenester, der man erstatter f.eks. pin-koder.
- internett-bank tjenester
- internett tjenester. Herunder E-handel. Digitale signaturer benyttes for å autentisere at begge parter er den de gir seg ut for å være. Men siden digitale signaturer ofte er basert på passord, ligger det en mulighet for at uautoriserte kan signere med din signatur.
- Offentlig etat. For eksempel søkeverktøy i ansiktsdatabaser for politiet osv. Ansikter er gunstig fordi det er slik vi mennesker gjenkjenner hverandre, og det er derfor lett å få sosial aksept for teknologien (i motsetning til for eksempel fingeravtrykk hvor man føler seg som en kriminell....).

2.2 Omgivelsene

Vi skal benytte dette systemet installert i første omgang på en *Linux*-box. På denne datamaskinen skal man ha en skjerm, tastatur og mus. I tillegg til dette skal man benytte et web kamera, som i vårt tilfelle er tilknyttet *USB* porten. Hele systemet skal knyttes til det vanlige norske strøm nettet som har en spenning på ca 220-230 V.

2.3 Systemets brukere

Potensielle brukergrupper kan være:

* Adgangskontroll basert på ansiktgjenkjenning. Herunder banktjenester, politi, rettsvesen og andre lignende behov. Brukerne i dette segmentet skal ikke ha store datakunnskaper, men med normale instruksjoner og kursing skal de lett benytte systemet.

* Brukergrensesnitt for personer med funksjonhemminger. Her skal man forutsette at brukeren trenger minimale instruksjoner med at man skal kunne kommunisere med systemet uten noen som helst forkunnskaper.

* Spill/Underholdningsystem for brukeren. Her skal man forutsette at brukeren har ingen eller vanlig datakunnskaper og enkle bruksanvisninger skal være nok.

* Videre utviklingsprosjekter. Dette er erfarne brukere og utviklere. For disse skal man lage en god dokumentasjon som skal beskrive systemet og alle dets funksjoner utgående.

* Videre utviklingsprosjekter, der man kan vha systemet implementere nye algoritmer og test hvordan disse fungerer på en utvalgt datasett.

2.4 Funksjon

Vi har en modul der man mottar et bilde fra web kameraet (et detektert ansikt fra *BIRI1*) og man skal sette dette bildet inn i databasen sammen med opplysninger om den personen dette bildet er av. Disse opplysningene gis i editerings felter.
Vi har en modul der man kan se på hva som finnes i databasen, en *browse* modul. Her kan man bla seg frem og tilbake i databasen og lese opplysninger om personene som er lagret i databasen. I denne modulen finnes søke muligheter der man kan søke frem en person ut i fra for- og etter navn. I modulen kan man i tillegg endre og slette poster i databasen.
I en siste modul har vi selve logikken for å gjenkjenne en person som kamera har fokus på. Her skal man presentere et resultat sett som er rankert etter hvilken grad man gjenkjenner personen ut i fra det som ligger i databasen.

2.5 Operasjon

Ved normal operasjon skal systemet sørge for at man skal ta imot og streamer data fra et web kamera og gjøre en et utklipp av ansiktet til personen som sitter foran kameraet. Dette er det *BIRI1*-prosjektet som gjør. Dette utklippet benyttes til å gjøre operasjoner mot databasen som inneholder bilde-poster av personer. Man skal presentere og rangere personene som det blir gjort godt treff på. Man skal kunne navigere i databasen(*browse*) og gjøre innsetting, endring

og sletting av data fra databasen. Man skal ved gjenkjenning ved hjelp av tale og animasjons delen, *BIRI3*, kommunisere med personen som er gjenkjent.

Systemet skal automatisk og kontinuerlig foreta deteksjon av ansikter foran kamera og man skal gjøre utklipp av treffet.
Feil der man ikke kan detektere et ansikt foran kamera skal detekteres slik at man ikke sender et tomt bilde til gjenkjenning. Dette skal fanges opp med feil sjekker og man skal komme tilbake til normal tilstand der man søker etter ny deteksjon. Feil som skjer mot databasen skal detekteres gjennom feil sjekker og man skal sikre seg mot feil bruker-interaksjoner og systemet skal komme trygt over i normal tilstand ved slike anledninger.

Vi setter ingen bastante krav til at systemet skal fungere prikk fritt fordi vi benytter teknologi (*Qt* og *MYSQL++*) som er ny for prosjekt gruppe medlemmene.

2.6 Aspekter omkring livssyklus

Det er naturlig ikke mulig å lage et komplett og feilfritt system i dette hovedprosjektet. Hvis systemet skal kunne ha en livssyklus som går utover skoleprosjektet, må vi legge til rette for videreutvikling.

Når prosjektet er ferdig, håper vi å kunne bringe det videre til oppdrags giver, slik at han kan administrere videre utvikling og utvidet bruk av systemet i undervisnings/forsknings formål. Samt at vi håper at man vil legge koden ut på Internett (http://prosjekter.hig.no/biri2 `http://prosjekter.hig.no/biri2`) slik at utviklere som er interessert kan laste ned det eksisterende prosjektet.

Utvikling i *Open Source* kjennetegnes ved at kildekoden gis ut under en egnet lisens, tilpasset formålet med programvaren. Felles for de klassiske lisensene som skal brukes er at alle har rett til å gjøre forandringer i kildekoden og distribuere den videre.

Dokumentasjon av koden skal være tilgjengelige i flere forskjellige formater (*HTML*, *latex*, *rtf* etc.).

3 Gjenkjennings algoritmer

3.1 Innledning

Maskin gjenkjenning av ansikter fra et stillende stående bilde eller i en video sekvens er et forskningsområde som spenner seg over flere disipliner slik som bildebehandling, mønster gjenkjenning, datasyn og neurale nettverk. Ansikts gjenkjennings teknologi har mange kommersielle og rettsvesen-aktuelle bruksmuligheter (nevt i kapittel 2).

Selv om mennesker tilsynelatende ser ut til å klare å skille ansikter ut i fra en bråkete og forstyrrende bakgrunn, er det å bruke en datamaskin til dette, en mer vanskelig oppgave.

Et tenkt problem i dette området kan formuleres slik: Gitt en video eller et still bilde av en scene, identifisere personen i scenen ved hjelp av bilder som ligger lagret i en database. Problemet her ligger i å segmentere ansiktet fra scenen, uttrekking av egenskaper i ansiktet for gjenkjenning. Det helhetlige BIRI-prosjektet omfatter disse områdene. I *BIRI1*-modulen ligger det å segmentere ansiktet fra en scene og vi i BIRI2 prosjektet har som oppgave å identifisere personen som det er trekt ut ansiktet av.

I de siste årene har ulike fagområder forsket på området, der de har fokusert på ulike retninger innen for området. Innen psykologi- og medisin har det vært forsket på områder som har vært konsentrert på ansiktgjenkjenning hos mennesker (egenskapen å gjenkjenne), som er gjort ved hjelp av holografisk analyse eller ved å trekke lokalt ut karakteristikker hos en persons ansiktet. Det forsket på hvordan nyfødte gjør dette og hvordan voksne mennesker organiserer og lagrer ansikter i hjernen. Alt dette har vært grunnlaget for utvikling av forskellige algoritmer for gjenkjenning, identifisering og lagring av ansikts-bilder.

De siste årene har man økt aktiviteten i området der man trekker ut øyer, munn osv. i et ansikt. Og områder som design av statistiske og *nevrale nettverk* klassifiseringer for ansikts gjenkjenning. Klassiske konsepter som Karhunen-Loeve transformasjon baserte metoder, singulær verdi dekomposisjon og som nevnt over *nevrale nettverk*, har blitt brukt[14].

I denne oppgaven vil vi ikke implementere alle metoder, men prøve kun metoder som fungerer tilfredsstillende ut i fra at vi i dette prosjektet ikke har muligheten til å ta tak i et så stort område som denne disiplinen omfanger. Men heller lage en fungerende applikasjon for videre utvikling og testing av mer kompliserte teknikker.

3.2 Ansikts gjenkjenning

Beskiver anvendte metoder og teorier i et ansiktgjenkjennings system.

3.2.1 Segmentering

For at man skal kunne gjenkjenne et ansikt må man segmentere ansiktet ut i fra en scene. Dette er av *BIRI1* modulen handler om.

Etter en enkel forklaring kan dette foregå slik: En utjevnet-versjon av orginal bildet er søkt gjennom for å finne kanter for kan forme en omkrets rundt et objekt, f.eks. et hode. Den detekterte kanten er deretter projeksjonert tilbake til bildet, og et fin-søk er lokalt gjennomført i detter omrisset. Her finner man lokasjon for øyer, nese og munn (forhåpentligvis).

Det er å legge merke til at for å segmentere et ansikt så benytter BIRI1-modulen (de trekker ut ifra en video-sekvens) en teknikk der de har prosessert et standard ansikt ut i fra en mengde ansikter, kalt *eigenface*, som de søker gjennom et bilde med og undersøker den *Euclidske* avstanden mellom test bildet og dette *eigenface* bildet.

3.2.2 Dra ut spesielle trekk i et ansikt

Her trekkes ut spesielle trekk i et ansikt for er representert i *eigenpictures*. Et mye brukt metode er noe som kalles *egenvektorer*(referert som "*eigepictures")*. Her kan alle bilder representeres som en vektor der hver verdi i denne *egenvektoren* er innslaget av *eigenpicture*. En vekted kombinasjon av dette gjenskaper bildet [14]. Hvis man bruker et økende antall *eiegenpictures*, man vil oppleve en mer nøyaktig tilnærming til bildet.

3.2.3 Gjenkjenning

I selve gjenkjennings delen er det brukt mange forskjellige metoder.

En statistisk tilnærming er mye brukt der man bruker *eigenpictures* for ansikts deteksjon og identifisering. Som beskrevet i avsnittet over kan hvert bilde beskriver som en vektor med vektede verdier av hver av disse *eigenfaces*. Vektene er oppnådd ved enkelt å prosjektere bildet inn i *eigenface* komponentene ved en enkelt indre-produkt-operasjon. Når et nytt test bilde av en person skal brukes, blir bildet representert som en vektor med vekter av *eigenfacees* (*egenvektor*). Identifikasjonen av bildet i databasen er gjort ved at man kalkulerer den *Euclidske* avstanden mellom bildene i databasen og test bildet og det som er nærmest kommer nærmest en match mot test bildet. Dette vart gjort i et forsøk beskrevet i [14], der de brukte en database med 2500 ansikter av 16 personer med forskjellig hode rotasjon, 3 forskjellige hode størrelser og 3 forsjellige lyssettinger. De rapporterte 96% korrekt klassifikasjon med lyssetting variasjoner, 85% med rotasjons variasjon og 64% med størrelse variasjonen.

En annen metode som er brukt kalles *nevral nettverk* tilnærmelse. Metoden innebærer at man undersøker forskellige attributter i et ansikt f.eks. øyebryntykkelse, nase-størrelse, munn-størrelse, kinn radius osv . *Nevrale-nettverk* går i sekvenser der man har en slags lære prosess og en klassifikasjons(testing) prosess. Dette går i en slags rundgang slik at man "lærer" av data som kommer inn i en "ikke-lineær" måte. I vår sammenheng kan det sammenlignes med at nye data kommer inn i databasen og man "lærer" nye karakteristikker av de nye bildene av ansikter. Man kan da regne ut *korrelasjon* mellom test bildet og de som ligger i databasen for å finne ut den som ligger nærmest test-bildet fra databasen.

En annen tilnærmelse mye brukt er *karakteristikk*-utdragning. Her lagres karakteristikk punkt som blir detektert ved å bruker *Gabor* filter. Disse kan trekke ut karakteristikker i ulike retninger. Dette kan f.eks. lagres på datafiler for hvert bilde. Dette reduserer lagrings kravet til databasen. Typisk så får man 35-45 punkt for hvert bilde. Identifikasjons prosessen gjør bruk av informasjonen i den topologiske grafiske representasjonen av karakteristikk punktene. Man regner på to kostnader: topologisk og likhet. Så blir den totale kostnaden for dette utregnet som bestemmer hvor nært man er det input bilde man skal teste mot.

3.2.4 Hva har vi implementert av algoritmer

Vi hadde ikke som mål å implementere kompliserte algoritmer for identifiseringen. I og med at vi antok at vi i fra BIRI1-modulen tok i mot et bilde som var segmentert fra en scene og der ansiktet var funnet. Vår jobb slik vi anså den var å implementere en enkel algoritme og ikke eksperimentere for mye med de ulike algoritmene. Men selve teorien rundt de forskjellige gjenkjennings teorien vart undersøkt da vi i starten av prosjektet ville undersøke hvilken algoritmer og metoder vi selv skulle implementere.

Vi satset på at vi skulle ha et testbilde og og bilder i databasen som det skulle testes mot. Man endret størrelsen på de bildene i databasen til en 20x20 pixel bilde som skulle representere det vi noe tvilsomt kalte *egenvektor*. Denne *egenvektoren* vart altså 400 lang og representerte de 400 gråtone verdiene i dette 20x20 bildet.

De to algoritmene vi implementerte var: **L2Metric** og **CovarMetric**.
L2Metric kalkulerte den *Euclidske-avstanden* mellom test bildet og de som ligger i databasen ved hjelp av funksjonen:

I_{ij} er pixel verdien for input bilde. T_{ij} er pixelverdi for test bildet i databasen. Den *Euclidske* avstanden mellom de to bildene blir:

$d_{bE} = \sum_{ij}(I_{ij} - T_{ij})^2$

CovarMetric kalkulerte *korelasjonen* mellom test bildet og input bildet ved hjelp av funksjonen:

Distansen er definert som $d_{bC} = 1 - c(I, J)$ hvor $c(I, J)$ er *korrelasjonen koeffisienten* mellom input vinduet I og test vinduet T som er definert som

$$c(I, J) = \frac{\sum_{ij}[(I_{ij} - I)(T_{ij} - T)]}{\sqrt{\sum_{ij}[I_{ij} - I]^2 \sum_{ij}[T_{ij} - T]^2}}$$

hvor I og T er gjennomsnittlig pixel verdi i test -og input -vinduene.
Ved siden av disse algoritmene implementerte vi to forskjellige pre-prosesseringer som kunne velges før man brukte de ovenfor nevnte algoritmene for gjenkjenningen. De to var: **subtraksjon av av beste passende lineære intensitets planet** og **histogram utligning**. Dette gjorde vi for at vi skulle kunne få lov til å se hvor viktig preprosseseringen hadde å si når vi brukte bilder som va med forskjellig lysstyrke og med forskjellig kontrast.

4 Analyse

4.1 Metoder

I analysen skulle vi finne frem til hvilke problemer/oppgaver systemet er tenkt å løse, og hvilken funksjonalitet som må tillegg systemet for å til for å størst mulig grad fylle de behov og forventninger oppdragsgiver har. For en kunde vil det å gå i innkjøp av et nytt datasystem være en betydelig investering. Og alle av oss har vel hørt om dyre feilslåtte utviklingsprosjekter, det er derfor helt avgjørende at kundens behov, og krav til systemet blir spesifisert nøye slik at man i minst mulig grad får store overraskelser langt ute i prosess forløpet. Kunden må derfor inkluderes ikke bare i analyse fasen, men gjennom hele utviklings prosessen. Det meget viktig å gjøre en god jobb med kravspesifisering og analyse av systemet, for her legges grunnlaget for det videre arbeidet med utviklingen. Gjør man en god jobb i analyse fasen vil det kunne kaste godt av seg i from av å lette det videre arbeidet med design og koding.

Fordi vi valgte *RUP* som utviklings modell som baserer seg på *Objekt Orientert* tankegang ble det naturlig å bruke *OOA*. Vi hadde også del forkunskaper innen *OOA* og *OOD*, men mest på det teoretiske plan, da dette utgjorde hoved tyngden i fagene systemutvikling 1 og 2 ved Høgskolen i Gjøvik.

4.2 UML modellering

UML står for Unified Modeling Language ble introdusert i 1997. *UML* har raskt utviklet seg til å bli det dominerende modellerings språk innen objekt orienterte systemutviklingsmetoder. *UML* brukes for å beskrive arbeidsprosessene og gir oss et sett av teknikker for å bygge systemmodeller på ulike detaljnivåer. Teknikkene dekker *OOA* og *OOD*. Modellene som brukes skal i første rekke gi et forenklet bilde av virkeligheten slik at man kan danne seg en oversikt for så kunne jobbe videre for å få frem detaljeringen rundt systemet. I design fasen ser man mer konkret på hvilke løsninger/metoder som bør benyttes for å ta hånd om kravene til systemet.
I analyse fasen tok vi i bruk i to teknikker :**Use Case modellering** og **Konseptuell modell/klasse diagram**.

* *Use Case* modellering tar sikte på å få frem funksjonaliteten systemet skal yte, samt si noe om grensene mellom systemet og omgivelsene (System Boundry).

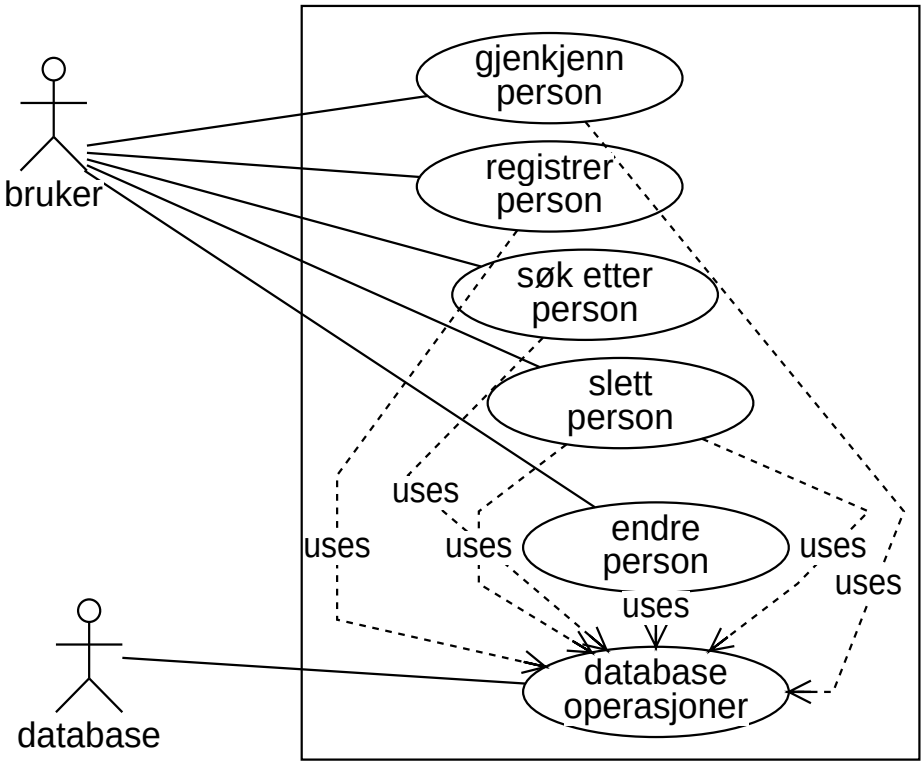
* Konseptuell modell skal ut fra virkeligheten gi et forslag/oversikt over aktuelle klasser som bør inngå i løsningen.

Vår *Use Case* modell har sitt utgangspunkt i krav spesifikasjonen der det går klart frem hvilke krav som stilles til funksjonalitet og operasjonalitet til systemet.

4.3 Definering av Use Case

I seksjonen skal vi klare gjøre litt rundt Use Case generelt, samt gå litt mere inn på de *Use Case* som er definert i vårt system.

se figur 4.1



figur *4.1 Use Case (bruksmønster) digram for systemet*

Strekmennene i *UML* representer brukere/aktører i systemet, mens sirklene representerer de ulike operasjonene aktøren kan iverksette. Det litt spesielle med vårt *Use Case* diagram er den utstrakte bruken av "uses" operanden. Det denne forsøker å illustrere er hvordan de ulike operasjonene aktøren i verksetter, videre gjør seg nytte av database funksjonalitet indirekte. Vi har også valgt å trekke selve databasen ut som en egen aktør da den kan ses på som en selvstendig del av systemet inneholdende sine egne operasjoner/funksjonalitet. Under følger en mer detaljert beskrivelse av de enkelte *Use Casene* med krav og aspekter rundt disse.

4.3.1 Funksjonelle krav til Use Caset "registrer person"

Beskrivelse:

Når en bruker setter seg ned ved ved maskinen foran kameraet og velger registrer-valget fra menyen linjen i hoved vinduet, skal det tas et bilde av brukeren ved hjelp av kameraet tilknyttet maskinen. Bilde som tas skal dukke opp i nytt vindu. Inne i dette vinduet skal videre brukeren kunne taste inn sitt fornavn, etternavn og en kommentar til bildet. Brukeren gis mulighet til å kunne foreta en ny bilde fangst vis han/hun ikke er fornøyd med resultatet av den første. Videre kan brukeren trykke på en registrer knapp for å foreta selv innleggingen mot databasen. Vinduet og operasjonen avsluttes med en avslutt knapp.
Tilleggs-funksjonalitet:

Vi har lagt opp til at vår del av det totale *BIRI* systemet skal kunne kjøres uavhengig av resten, derfor har vi lagt inn ekstra funksjonalitet slik at bilder også skal kunne lastes opp fra fil.
Operasjonelle-krav:

Maksimal respons tid 3 sekunder. Ved feil fra bruker eller systemet side skal det gis fornuftige feilmeldinger i form av informasjon/advarsel vinduer. Samt melding om at registrering er gjennomført dersom alt gikk greit.

4.3.2 Funksjonelle krav til Use Caset "søk person"

Beskrivelse:

Når brukeren velger "søk etter person" fra meny-linjen, skal et nytt vindu komme opp på skjermen. Her skal bruker kunne foreta søk etter personer i databasen på fornavn og/eller etternavn. Ved treff i databasen ut fra søket som ble gjennomført, skal treffene listes ut på skjerm i form av bilder med tilhørende bilde identifikasjon. Er treffet for søket på større enn 10 personer gis brukeren mulighet for å bla (browse) resultatet vha. knapper for frem og tilbake. Vinduet og operasjonen avsluttes med en avslutt knapp.
Operasjonelle krav:

Maksimal responstid 5 sekunder.Ved feil fra bruker eller systemet side skal det gis fornuftige feilmeldinger i form av informasjons/advarsel vinduer.

4.3.3 Funksjonelle krav til Use Caset "endre data om person"

Beskrivelse:

Når brukeren ønsker å gjøre oppdateringer på en person som er registrert i databasen gjøres det ved at "endre data om person" velges fra meny linjen i hoved vinduet. Da vil det dukke opp et nytt vindu på skjermen, vinduet vil inneholde bilde av de 10 første personene som er registrert i databasen. For at brukeren skal da kunne finne personen han/hun ønsker å redigere registrerte data om, kan man enten bla seg frem steg for steg i databasen vha. frem og tilbake knappene. Eller man kan foreta et søk som beskrevet i *Use Caset* "søk person". Når man har funnet frem til personen man ønsker å redigere klikker man på bildet/bilde identifikasjonen til vedkommende. Da vil info tilhørende denne personen dukke opp i tekst felter, og vil kunne redigeres. Når man har utført ønsket redigering av den valgte personens data trykker man på oppdater i endringen blir registrert i databasen. Vinduet og operasjonen avsluttes med en

avslutt knapp.
Operasjonelle krav:

Maks responstid 5 sekunder.Ved feil fra bruker eller systemet side skal det gis fornuftige feilmeldinger i form av informasjon/advarsel vinduer.

4.3.4 Funksjonelle krav til Use Caset "slett person fra databasen"

Beskrivelse:

Når brukeren ønsker å utføre sletting av en person som er registrert i databasen gjøres det ved at "slett person far databasen" velges fra meny linjen i hoved vinduet. Da vil det dukke opp et nytt vindu på skjermen, vinduet vil inneholde bilde av de 10 første personene som er registrert i databasen. For at brukeren skal da kunne finne personen han/hun ønsker å slette registrerte data om, kan man enten bla seg frem steg for steg i databasen vha frem og tilbake knappene. Eller man kan foreta et søk som beskrevet i *Use Caset* "søk person". Når man har funnet frem til personen man ønsker å slette fra databasen klikker man på bildet/bilde identifikasjonen til vedkommende. Da vil info tilhørende denne personen dukke opp i tekst felter, samt at bilde vil kunne slettes ved å trykke på slette-knappen. Da vil all data om valgt person slettes fra databasen. Vinduet og operasjonen avsluttes med en avslutt-knapp.
Operasjonelle krav:

Maksimal responstid 5 sekunder.Ved feil fra bruker eller systemet side skal det gis fornuftige feilmeldinger i form av informasjon/advarsel vinduer.

4.3.5 Funksjonelle krav til Use Caset "gjenkjenn person"

Beskrivelse:

Når en bruker setter seg ned ved ved maskinen foran kameraet og velger "gjenkjenn person" fra menyen linjen i hoved vinduet, skal det tas et bilde av brukeren ved hjelp av kameraet tilknyttet maskinen. Dette bildet blir sammenlignet mot bildene som er registrert i databasen, og det bilde som systemet finner ligner mest på input bilde blir det hentet ut data om. Data om bilde det ble "treff" på sendes til delen *BIRI3* har implementert, og det starter en animasjon samt talegenerering for å fortelle brukeren hvem systemet tror han/hun er. Inne i dette vinduet skal videre brukeren kunne taste inn sitt fornavn, etternavn og en kommentar til bildet. Brukeren gis mulighet til å kunne foreta en ny bilde fangst hvis han/hun ikke er fornøyd med resultatet av den første. Videre kan brukeren trykke på en registrer-knapp for å foreta selv innleggingen mot databasen. Vinduet og operasjonen avsluttes med en avslutt-knapp.
Tilleggs funksjonalitet:

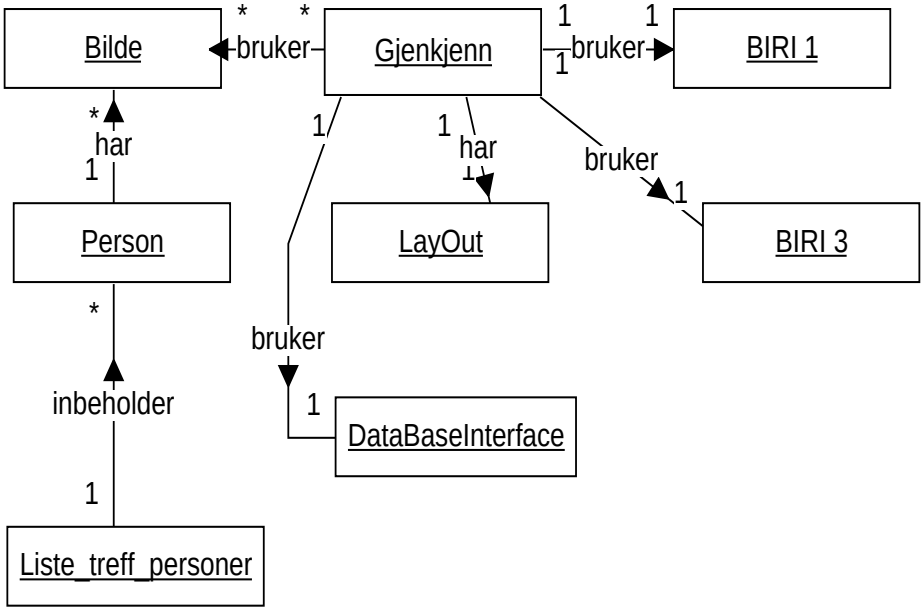
Vi har lagt opp, som nevnt ovenfor at vår del av det totale Biri systemet skal kunne kjøres uavhengig av resten, derfor har vi lagt inn ekstra funksjonalitet slik at bilder også skal kunne lastes opp fra fil. Og at gjenkjenningen kan skje uten at det startes noen animasjon eller talegenerering. Hvis systemet kjøres uten *BIRI3* delen vil gjenkjenningen foregå på en litt annen måte sett fra brukerens ståsted. Selve initieringen av gjenkjennings prosessen blir lik, man kan enten

laste opp bilde fra kamera eller fra fil og bilde blir sjekket mot bildene registrert i databasen for å finne det med størst likhet i forhold til input bilde. Forskjellen kommer først til syne i det resultatet av gjenkjenningen skal presenteres. Uten *BIRI3* delen av systemet, vil resultatet av gjenkjeningen bli presentert i et nytt vindu inneholdende bilde av de 10 mest like personene i forhold til input bilde, rangert etter likhet. Det nye vinduet vil også inneholde knapper med funksjonalitet for å laste inn et nytt bilde, enten fra kamera eller fil. For så å kunne kjøre en ny gjenkjennings operasjon på det bilde. Vinduet og operasjonen avsluttes med en avslutt knapp.
Operasjonelle krav:

Maksimal responstid 10 sekunder.Ved feil fra bruker eller systemet side skal det gis fornuftige feilmeldinger i form av informasjon/advarsel vinduer.

4.4 Konseptuell modell/klasse diagram

Som såvidt nevnt under delen om UML modelleringen, er hensikten med konseputell modell/klasse diagram å få frem kandidater til egne klasser ved å se på de omgivelser systemet skal opptre i (finne konsepter). Det er viktig å presisere forskjellen mellom konseptuellt klasse diagram og software klasse diagram. Konseptuelle klasse diagram tar direkte utgangspunkt i ting fra virkeligheten og inneholder ikke noen metoder/funksjoner. Det er utviklet egne strategier for hva man skal se etter i virkeligheten for å lette utarbeidelsen av konseptuelle modeller. Analyse av de de funksjonelle og operasjonelle kravene som er definert i kravspesifikasjon er en måte. På figuren 4.2 er det synlig gjort det vi kom frem til.



figur 4.2 Konseptuell modell/klasse diagram

5 Design

Trekker her ut de viktigste aspekter fra UML modelleringen som følger med som vedlegg C.

5.1 Prosessen

Vi eksperimenterte med ulike type UML diagrammer. Vi bestemte oss som nevnt i kap.4, tidlig i prosjekt perioden at vi ville utforske området *Objekt Orientert Analyse og Design*. Vi ville benytte metoden som ligget i modellerings rammeverket *UML*, fordi vi vet at dette er et rammeverk som benyttes under nær sagt alle profesjonelle *objekt orienterte* utviklings prosjekter.

Etter en iterativ utviklings syklus er det mulig å flytte til design fasen, etter at analyse dokumentene er ferdige. Gjennom dette steget blir, den logiske løsningen basert på det objekt orienterte paradigmen, utviklet. Men man kan hele tiden ta tilbake-hopp i prosessen for å addere eller subtraktere ting som dukker opp.

Å designe vil si å legge planer for hvordan den fremtidige situasjonen skal være. Analysen forteller hva systemet skal gjøre, men ikke hvordan der bør gjøres. I design delen vil vi prøve å konstruere systemet for å møte kravene til systemet så godt vi kan.

Tidlig i prosessen hadde vi som mål og lage en god *GUI* design, database og enkel *algoritme* for å gjenkjenn en person. Ettersom prosjektet skred frem, og noen av prosjekt medlemmene fattet mer og mer interesse for selve den gjenkjennings prosessen og bilde prosesseringen, fant man ut at man skulle gjøre designet så generelt slik at man enkelt kunne implementere nye algoritmer og preprosesserings rutiner som kunne testes ut. Det videre designet av løsningen bar preg av dette.

5.2 GUI design

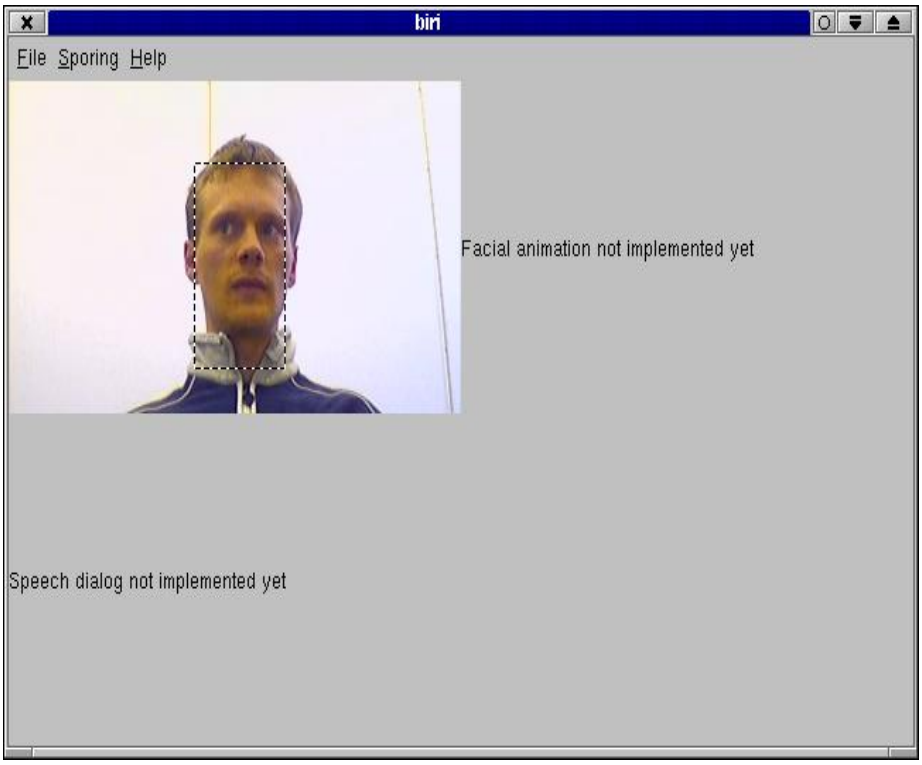
5.2.1 Overordnet GUI design

BIRI2 skal være en *SDI* applikasjon. Det vil si at hver del har hvert sitt vindu med sine egne knapper. Selve database browsing modulen, gjenkjennings modulen og registrerings modulen blir åpnet i nye vinduer (motstykket til dette er *MDI*, hvor man har et hovedvindu som inneholder alle deler og dokumenter). En del synes *SDI* er rotete, men fordelen med det er at det gir muligheten til å jobbe på flere desktop'er eller skjermer samtidig.

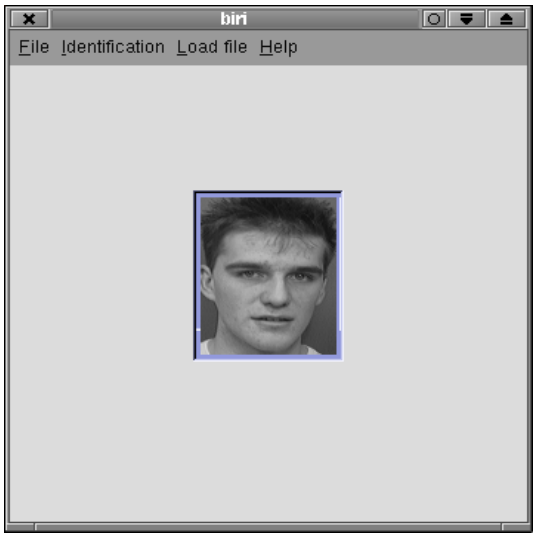
5.2.2 Design for det felles BIRI prosjektet

Vi bestemte oss i hele BIRI gruppen for å lage et hoved vindu (se fig.5.1) som skulle være så enkelt som mulig. Man skulle basere hele meny systemet på at alle valgene skulle ligge i en meny linje øverst i applikasjonen som følger vanlig standarder for f.eks. tekst behandlings applikasjoner.

Dette gjorde vi for å skape en enkel navigasjon mellom de ulike modulene i *BIRI* prosjektet (mellom *BIRI1*, *BIRI2* og *BIRI3*). En meny linje skal inneholde de enkle meny valgene som "File" og "Help" som har de standard rutinene som du finner i applikasjoner. Hoved vinduet for kjøring uten kamera utviklet ene og alene for biri2. (se fig.5.2)



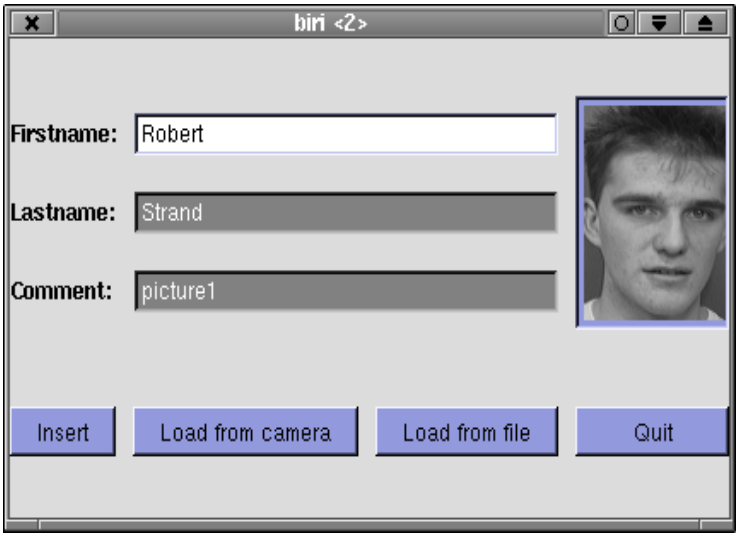
figur 5.1 hovedvindu for felles BIRI applikasjon



figur 5.2 hovedvindu for BIRI2 applikasjon

5.2.3 Design for registrering i databasen

Vi bestemte oss for at denne ikke skulle være så komplisert. Enkle editering felter for fornavn, etternavn og en kommentar ble laget. Sammen med et bilde utklipp som viste bildet som skulle registreres(se fig.5.3). I denne delen skulle det bildet som var fanget i *BIRI1* modulen sendes med til denne rutinen, men at man samtidig hadde muligheter til å laste opp bilder fra fil eller få et nytt utklipp av personen foran kameraet hvis det orginale utklippet ikke var tilfredsstillende.



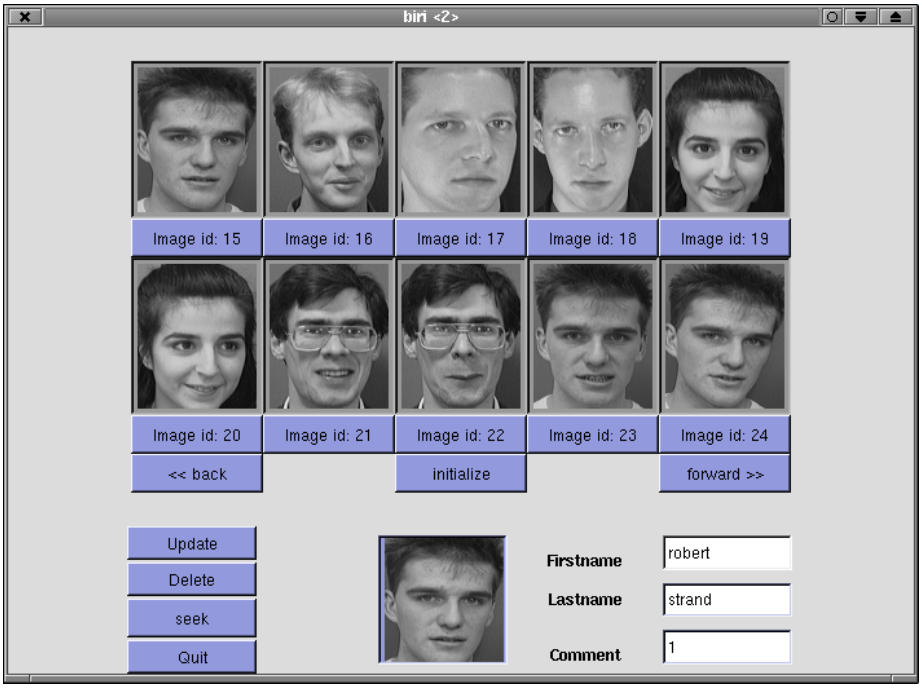
figur 5.3 bilde av registrering i databasen

5.2.4 Design for browsing i databasen

Denne delen av *GUI* programmeringen ble mer komplisert. Vi måtte bestemme hvilken funskjonalitet vi ønsket skulle være med i denne delen. Vi fant fort ut at i denne delen ønsket vi å ha muligheten til å bla frem og tilbake i databasen, gjerne 10 poster om gangen på skjerm. Her skulle det være mulig å klikke på et bilde som ble vist, og man skulle kunne lese de registrerte opplysninger om denne personen og editere disse. Ved siden av dette skulle det være en slette mulighet for å slette den posten som det var klikket på. Til slutt skulle man ha muligheter til å søke etter en person ut i fra fornavn og etternavn.

Man baserte seg fort på at bildene av personen skulle listes ut i 2 linjer med 5 bilder på hver med en trykk knapp under slik at du kunne velge dette bildet for å se på det.

Resten av *GUI*'en skulle basere seg på knapper og enkle editerings felter. Knappene skulle samles sammen slik at man enkelt skulle kunne finne frem til riktig funksjonalitet (se fig 5.4).



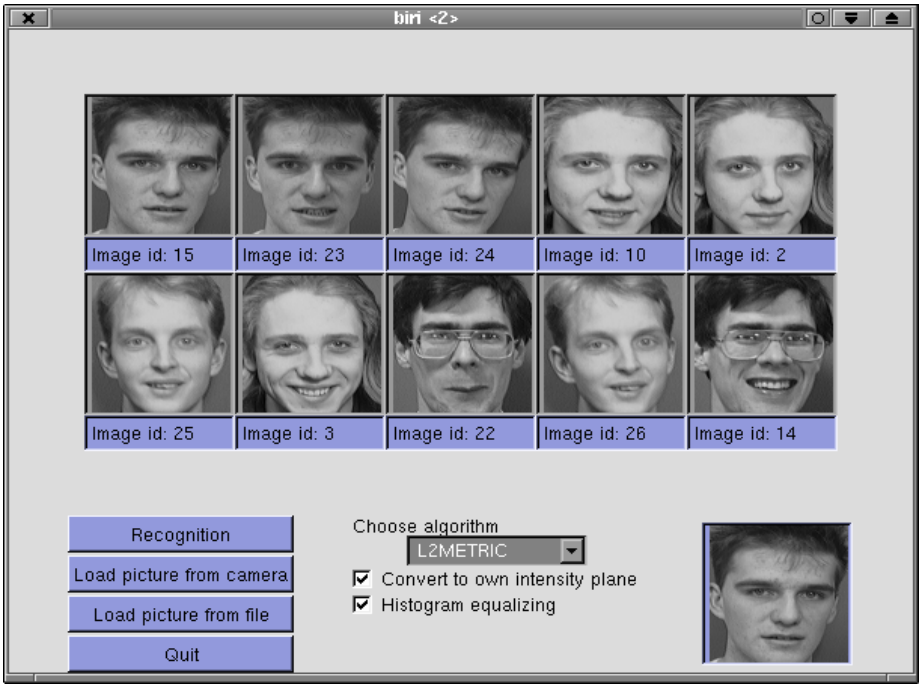
figur 5.4 figur av browsing av databasen

5.2.5 Design for gjenkjenning

Dette er den hoved funksjonen i BIRI2 løsningen. Her bestemte vi fort at vi skulle liste ut alle de kandidatene som kunne matche input bilde opp i rangert rekkefølge. Vi skulle liste de 10 mest matchende kandidatene på to linjer med 5 på hver. Man skal kunne velge i en *drop-down-boks* hvilken algoritme man ønsket å bruke og i *check-boxer* merke av hvilken preprossesering man ønsket å utføre før gjenkjenningen.

I dette vinduet (se fig.5.5) skulle man ha muligheter til å laste opp en ny billed fangst fra web-kameraet eller man skulle laste opp et nytt test bilde av et ansikt fra fil.

Valgene for opplasting, gjenkjenning og avslutning av vinduet skal ligge i vanlige trykk-knapper.



figur 5.5 figur av gjenkjenningsdelen

5.3 Kode design

5.3.1 Generelt om klassediagrammer

Detaljerings graden i et klasse diagram har en tendens til å bli for stor og man mister oversikten og har en tendens til å bli unødig kompliserte. Det var tanke vi gjorde da oss da vi gikk fra analyse fasen til design fasen.

5.3.2 Om kode design

I utgangspunktet ønsket vi å lage en applikasjon med en enkel og god GUI, implementere database håndtering og lage noen enkle gjenkjennings algoritmer.

Vi ønsket å bruke den "three-tier architecture"[6] modellen for å skille mellom presentasjonen ut til bruker, mellom applikasjons logikken og lagringsmediumet. Dette mener vi er en god modell med bakgrunn i:

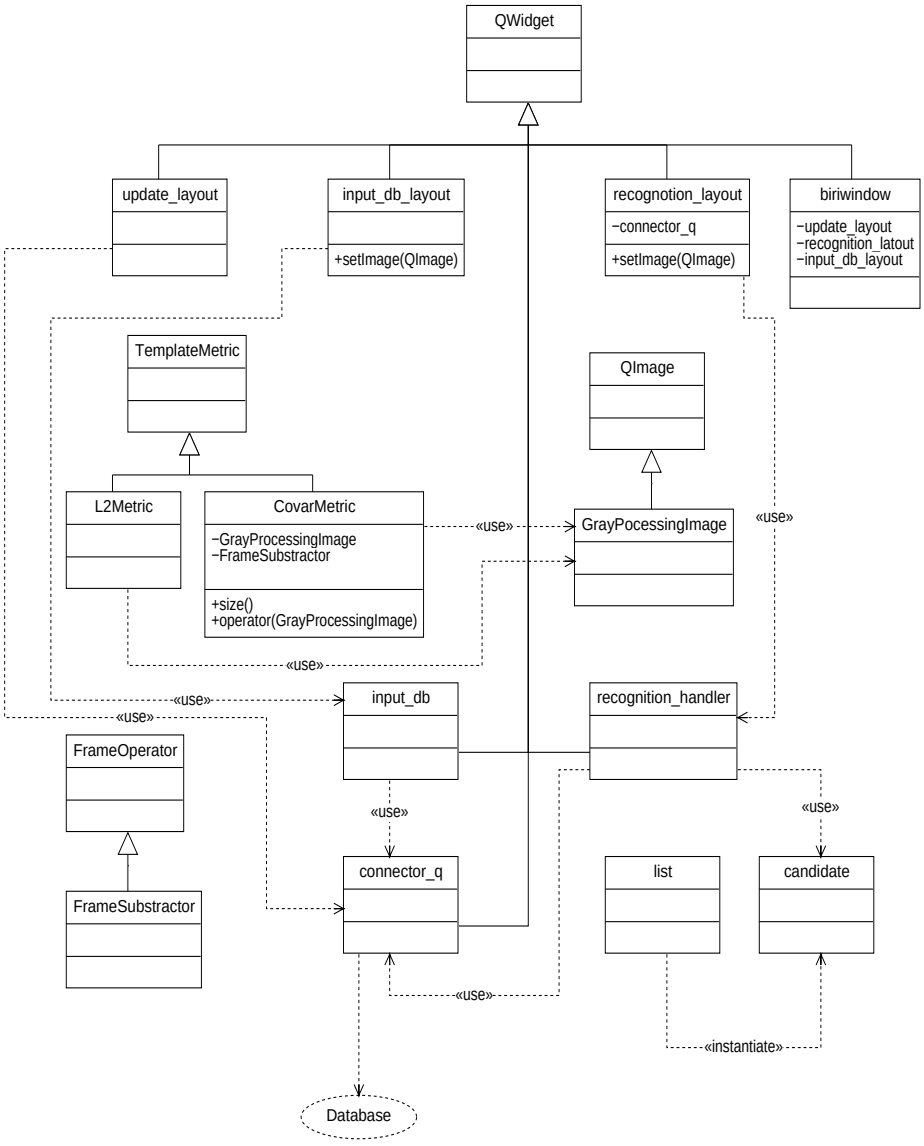
* Isolasjon av applikasjons logikken i separate komponenter som kan bli gjenbrukt i andre systemer.

* Distribusjon av lag på forskjellige fysiske dataprosseserende noder, og/eller forskjellige prosesser. Dette kan forbedre ytelsen og øke koordinasjon og deling av informasjon i et klient/server system.

* Allokering av utviklere til å konstruere et spesifikt lag, som f.eks. et team som arbeide kun på presentasjons laget. Dette støtter spesialisert ekspertise, og mulighetene til å kjøre parallelle utvikling.

5.3.3 Programmets overordnede struktur

Figur 5.6 viser et oversiktsdiagram for løsningen vi valgte for implementering av *BIRI2*. Den viser arv, og noe hvordan de ulike objektene henger sammen for å vise hvordan de bruker hverandre. Figuren viser ikke alle funksjonene til hver klasse (det hadde blitt en del), men kun nok til at de er beskrivende for klassens oppgaver. Denne løsningen bygger under den tanke gangen om å skille de forskjellig lagene fra hverandre og f.eks. bruke de på den måten som beskrevet i forrige punkt.



figur 5.6 Design/software klasse diagram

5.4 Generelt om design fasen og UML

I og med at vi bestemte oss for å benytte *RUP* utviklings rammeverket kan man ikke skille mellom de ulike fasene da dette er en iterativ modell. Hver fase i *RUP* består av sin lille *fossefallsmodell* som man itererer over (forklart i kap. 4). Men jeg kan beskrive hvilken modellerings verktøy man benytter ved design fasen. Viktig i denne fasen er å lage *system sekvens* diagrammer som beskriver

et konkret design for hvordan Use Case'ene blir realisert. Etter dette lager man *kollaborasjonsdiagram* for å delegere ansvar mellom klasser og beskrive hvordan designet blir. Tilslutt former man et *design klasse* diagram som fungere som en mal for hvordan software klassene kommer til å være (figur 5.6).

6 Implementasjon

6.1 Generelt

Ettersom utviklings miljøet vårt er *Linux* og vi skulle implementere en applikasjon som fungerer for alle forskjellige distribusjoner, valgte vi *Qt* som utviklingsverktøy. Dette ble valgt på grunnlag av at vi skulle lage en *OpenSource* applikasjon samt at den skulle være i størst mulig grad plattform uavhengig. En annen grunn for at valget landet *Qt,* er og at *api*'et er veldig på frammarsj og blir mye brukt av utviklere for *Linux* plattformen. Dette hadde vi en mening om at kunne hjelpe oss fordi mange engasjerte programmere som bruker det og at dokumentasjonen er bra.

Qt er et grafisk bibliotek laget av et norsk firma **Trolltech**, dette biblioteket baserer seg på *C++* programmeringsspråk. Så vårt valg av utviklingsspråk falt derfor naturlig nok på *C++*. Samt at *C++* og er et *objekt orientert* programmeringsspråk som er ofte benyttet, og vil hjelpe ved videre utvikling av applikasjonen ettersom mange er fortrolige med dette språk.

Når vi skulle ta i bruk database falt valget på *MySQL* database. Dette er fordi vi hadde benyttet oss av denne ved en tidligere anledning så vi hadde kjennskap til funksjonaliteten, og at dette er en veldig stabil og sikker datebase. Vi hadde imidlertid ikke benyttet den med C++ men dette ble løst ved hjelp av *API*'en *MySQL++* for kobling mellom *C++* kode og databasen vi opprettet. *MySQL* er i likhet med *Qt* også plattform uavhengig.

Ettersom vi kom til å få en del *header* filer og *cc* filer var vi nødt til å benytte en script for kompliering og linking, det ble komplisert å kompilere hver fil manuelt etterhvert og linke hele prosjektet sammen. Da benyttet vi oss av en applikasjon som heter *Tmake* utviklet av **TrollTech**. Vi laget en pro-fil som vi definerte hvilke filer og bibliotek som skulle kompileres og linkes, *Tmake* genererer da en *Makefile*(scriptet) som kjøres og da kompileres og linkes prosjektet med filer med de forskjellige bibliotekene og man ender med en eksekverbar binær fil.

Selve kodingen foregikk ved bruk av *emacs (www.emacs.org)* som er en stabil og bra tekstbehandler.

Når det gjelder dokumentasjon av kode valgte vi å benytte oss av *Doxygen (www.doxygen.org)* som er et dokumenterings program for *C++*. Det finnes og flere men denne gir oss store muligheter ved at den genererer dokumentasjonen i flere forskjellige formater, blant annet *Latex* og *html* noe som er oss til stor hjelp når dette skal publiseres på nett og ettersom vi valgte og utføre rapport skriving i *Lyx* (`www.lyx.org`).

Lyx er et kraftig verktøy for rapport skriving og har et mange muligheter så vi skal ikke gå så dypt inn i det her, men det må nevnes at den blant annet benytter *Latex* standard. God struktur funksjonalitet med kryssreferanse, innholdsfortegnelse osv. gjør det lett å utarbeide dokument med meget god struktur.Den gir og mulighet for å eksportere dokumentet i flere formater som feks *PortabelDokumentFormat(pdf)*, *Post Script.*
Det bør og nevnes at alle applikasjoner som er benyttet er lisens frie. (hvertfall

for privat bruk).

6.2 Standarder for koding

Prinsippet for kodingen og kommenteringen var at vi skulle gjøre dette så ryddig og forståelig for andre så vel som for oss selv, ettersom dette er en applikasjon som skal kunne bygges videre på. I forprosjekt rapporten antydet vi at vi skulle gå ut fra *GNU (www. gnu. org)* kodestandarden og følge de retningslinjene som er der, men vi gikk litt bort fra dette ettersom den viste seg å bli vel omfattende for vårt behov og er mer basert på *C* kode. Vi har allikevel en god del likheter, vi tok med oss de elementer fra *GNU* standarden vi fant mest nyttig for å oppnå ryddighet og orden i koden. Slik at de som eventuelt senere skal sette seg inn i koden har et godt utgangspunkt, og tilrettelegge for videre utvikling av vår kode.

Vi har her valgt å definere klasser og funksjoner i egne header-filer og gjøre bruk av disse i vanlige *C++* filer. Dette ble gjort bevisst for gjenbruk, ettersom klassene da kan brukes individuelt. Det finnes en hovedklasse som benytter alle de andre.

Når det gjelder selve kodingen og retningslinjer derom bestemte vi oss for gi klassene beskrivende navn og gi selve filen samme navn. Navnsetting på klassene og variable skulle inneholde kun små bokstaver og skilles med "_" for hvert nytt ord, lokale variable definert i de klassene de brukes i skulle også begynne med en "_".

Alle egen definerte slot'er begynner med "slot". Alle klasse- og filnavn samt kommentering forekommer på engelsk fordi vi kan tenke oss utviklere utenfor Norges grenser som ikke kan norsk....

Kommentering av klasser og funksjoner begynner med "//!" , dette for å benytte *Doxygen* til dokumentasjonen. Ytterlige kommentarer for forståelse av spesielle operasjoner i funksjonene gjøres uten "!". Dette for å ikke å få det med i dokumentasjonen.

Kort eksempel:

```
public class eksempel_kode {} //eksempel på klasse deklarasjon

int eksempel_variabel; //vanlig definert variabel

char _lokal_variabel; //lokal definert variabel

public slot {//eksempel på definering av slot
slot_eksempel;
}
```

6.3 Implementasjons valg og problem under implementasjonen

Vi bestemte som tidligere nevnt for å prøve å lage en implementasjon som skulle skille lagene i en applikasjon i følge den three-tier-architecture [6]. Dette synes vi har klart da man ikke ha annen kommunikasjon mellom de forskjellige lagene annet at de laget over kommuniserer med laget under, men ikke andre veien.

Vi fulgte og brukte aktivt det analysen og design dokumentet vi laget (som følger med som vedlegg *UML*). Vi mener selv at vi har truffet ganske bra med den modelleringen vi gjorde av selve strukturen i applikasjonen og vi har hele tiden hatt en revisjon av dette dokumentet ettersom man har gått fremover i utviklingen.

Selve implementasjon kan deles opp i 3 deler:

- *implementasjon* av gjenkjenning algoritmer
- *GUI implementering*
- database *implementering*

Implementasjon av gjenkjenning vart noe endret i forhold til hva vi trodde vi skulle gjøre da prosjektet startet. Dette fordi når vi begynte med prosjektet, trodde vi at oppdragsgiver hadde definert hvilken algoritmer som skulle defineres. Vel å merke hadde oppdragsgiver gjort en del undersøkelser omkring området (diverse papers), men vi fikk aldri definert de algoritmene vi mente lå ferdig når vi startet med prosjektet. I og med at algoritmene for selve gjenkjenningen er kompliserte og teoretiske, vi opplevde at vi ikke fant noen "fasit" for hvordan dette vanligvis blir utført. Vi måtte gjøre en del undersøkelser for å finne en del teori rundt området. Etterhvert så fant vi ut at vi skulle implementere enkle algoritmer som skulle fungere tilfredsstillende med gode bildefangster. I starten gjorde vi enkle tester av bilder mot hverandre der vi undersøkte om to bilder var identiske, dvs. undersøke *piksel* for *piksel*. Etterhvert så begynte vi med metoder der vi regnet ut *variansen* mellom to bilder og rangerte de etter dette. Men etter å ha konfrontert med Ivar Farup, benyttet vi noen metoder som han og oppdragsgiver, Erik Hjelmås, hadde brukt for å segmentere ansikter fra en scene. Nemlig å regne ut den *euklidske*-avstanden og *korelasjonen* mellom to bilder[17]. Ettersom disse metodene fungerte brukbart på et test sett av bilder vi hentet fra Olivetti Reasearch Laboratory `www.orl.co.uk`), bestemte vi for å bruke disse og heller konsentrere oss på andre områder i applikasjonen. Sammen med disse vart forskjellige pre-prosesserings metoder implementert etter ideer fra Ivar Farup.

GUI-implementeringen foregikk med *Qt* som har gått bra. Det er etterhvert gitt ut en del bøker innen dette, samt at det ligger mye dokumentasjon på **TrollTech** sin hjemmeside (`www.trolltech.com`). Vi hadde en del oppstarts problemer med dette, men etterhvert fikk vi nok kjennskap til at vi klarte å

gjøre bruk av en del som *Qt* inneholder. I vurderingene om hvordan man skulle ordne vinduene i applikasjonen bestemte vi oss som forklart i kapittel.5 for et *SDI* -interface. Et felles vindu vart implementert for det felles *BIRI*-prosjektet, men samtidig lagde vi en eget hoved vindu som skulle fungere kun for vårt prosjekt. Dette for å sikre oss hvis en situasjon skulle oppstå der vi ikke fikk integrert oss mot de andre prosjektene og å være en mulighet for å få testet applikasjonen ut. Det som var utfordrende i implementasjon i *GUI* delen var der vi skulle liste ut kandidatene som kom fra sammenlignings prosessen og selve database-browse vinduet. Her vart utviklingen av layout dominerte og relativt tidkrevende da *Qt* ikke følger de metoder som vi hadde vært borti i f.eks. *Java*. Database-browsingen var komplisert da man skulle ha mulighet til å bla seg frem og tilbake i databasen.

Database *implementeringen* var komplisert da vi som nevnt over brukte et ganske nytt og for oss ukjent *api*. Vi bestemte oss tidlig for å implementere databasen i *MySQL*, og vi ville finne en måte og programmere mot *MySQL* på med *C++*. Etter en del undersøkelser på nett fant vi at det var et prosjekt på gang hos *MySQL* for å skrive et *API* i *C++*. Men denne prosessen var ikke så langt kommet ennå. Med *MySQL* så finnes det et *C*-programmerings *api* som man kunne benytte, men vi ønsket på bruker *C++* i all vår kode. Derfor ville vi undersøke dette. En par av oss begynte å undersøke dette *api*'et og hadde en del prøving og feiling før vi klarte å skjønne hvordan man skulle bruke det. Et problem i starten var installasjon av *api*'et for det viste seg å ikke være en triviell oppgave (les kap.6.3.2). Men tilslutt løste dette seg. Neste problem var hvordan man skulle få lagret selve billed-data i databasen noe vi aldri hadde gjort før. Etter å ha gravd oss ned i en del litteratur særlig i [8] fikk vi en løsning på dette. Denne boken er noe av den mest omfattende innen *MySQL*, men alikevel så stod det ingen ting om *C++* api der, men heldigvis var dette ganske likt metoder som blir gjort i *C*. Største problemet i bruken av dette *api*'et var at dokumentasjonen på det var begrenset, men god hjelp fra interesse-listen *plusplus@lists.mysql.com* og Sinisa Milivojevic gjorde at vi kom i mål her.Vi gjorde en del test-programmer og etterhvert så forstod vi hvordan dette skulle brukes.

Det som vi la spesielt vekt på under implementasjon var at man enkelt skulle kunne videreutvikle systemet og legge til nye algoritmer. Dette var gjort på en slik måte at man lager nye klasser for nye algoritmer og lager en operator for sammenligning av to bilder etter den metoden man ønsker.

I algoritme designet hadde vi spesiellt nytte av den koden vi kopiert fra en test applikasjon som Ivar Farup og Erik Hjelmås benyttet jfr artikkelen de publiserte i [17].

6.4 Erfaring med bruk av utviklings verktøy

- C++

Vi har gjennom utviklingen av systemet fått satt vår kunnskap om C++ på

prøve. Vi fant fort ut at vår kunnskap var noe rusten, vi måtte til stadighet ta i bruk læreboka i [2].

- Qt

Vårt inntrykk med bruk av *Qt* var i starten delte. Vår bruk innebefattet i hovedsak noen klasser som gikk på layout,bildebehandling og behandling av binære data for å få til jobbing mot databasen i systemet. Det er særlig ved layout delen i Qt vi slet med, ettersom det å legge komponenter inn i de forskjellige typer mønster ikke ga forventet resultat. Vi i gruppen har tidligere jobbet med lignende problemstillinger under programmerings språket *Java* og synes layout håndteringen der var mer stabil og forutsigbar. Man kan ikke legge skylden direkte på Qt, det kan være at vi benyttet feil innfallsvinkel på problemet. Vi var ivrige etter å komme igang og se resultater. I ettertid ser vi at vi kanskje skulle brukt enda mer tid på litteratur rundt emnet.

Det må sies at den dokumentasjonen som er utarbeidet rundt *Qt* er veldig anvendelig. Den er oversiktelig og rik på eksempler. Vi benyttet oss og av *Qt* gruppene på nett. Her var det mye kunnskap å hente fra ressurssterke personer. Når vi tok direkte kontakt via mailinglist ble vi imponert over responsen. Det tok ikke mange timene før fikk gode forslager til løsninger, evnt. forklaring på funksjonalitet.

Etterhvert som vi opparbeidet oss mer erfaring i bruk av Qt og hvordan man jobber med løsning/implementering av konkrete oppgaver ble vi mer fortrolig med dets funksjonalitet. Nå i ettertid sitter vi igjen med et positivt intrykk, det har vært interessant å jobbe med Qt og se likheter mellom andre verktøy som er utviklet for implementering av *GUI* applikasjoner. Alt i alt må vi si oss fornøyd med *Qt*.

- MySQL

Som nevnt tidligere har vi i vår løsning benyttet oss av database mangeren *MYSQL*, da vi hadde kjennskap til denne.

Det vi imidlertid ikke hadde tidligere erfaring med var lagring av bilder i databasen. Selve installasjonen og oppsettet av MySQL databasen er av de tingene som skapte minst vansker for oss. Vi benyttet oss kun av de enkle/tradisjonelle operasjonene som select og insert.

- MySQL++

Som kjent benyttet vi oss av *MySQL++ API* for kobling mellom *C++* og *MySQL* databasen. Vi hadde tidligere gode erfaringer med kobling mellom *Java* og *MySQL* databaser ved hjelp av ODBC. Det oppstod problemer ved installasjon av *MySQL API*'en. Problemet lot seg imidlertid løse etter tips hentet fra *MySQL* mailinglist. Det som viste seg å være problemet var at RedHat 7.0 ble distribuert med en betaversion av gcc kompileren(2.96). Vi kjørte en patch til kompileren og problemet var løst.

- utvikling under LINUX

Når det gjelder utvikling under Linux har vi gjort oss erfaringer med at problemer kan oppstå mellom de forskjellige versionene av diverse applikasjoner, kompilatorer og api'er. Det viser seg imidlertid at de fleste problemer lar seg løse. Vi lar oss ikke slutte å bli imponert av den informasjons mengden og hjelpen som finnes på nett. I *OpenSource* utvikling har vi fått bekreftet at det finnes mange entusiaster med stor kunnskap som gjerne svarer på spørsmål.

7 Testing

7.1 Testing av systemet under utvikling

Ettersom vi har mange klasser og funkjsoner som er avhengig av hverandre måtte vi foreta tester av hver klasse etterhvert for å være sikker på at de fungerte som forutsatt. Vi testet og sammensetting av forskjellige klasser før alt ble satt sammen i den endelige *applikasjon*.

Disse testene ble utført for å se om klassene hadde *kompilerings* feil, ikke den store funksjonalitets testen. Etterhvert som vi testet flere og flere klasser sammen ble testene mer omfattende. Dette innebærer også en grundigere funksjonalitets test for å være sikre på at klassene fungerte bra før de ble knyttet sammen til hovedklassen. Vi brukte ingen debugger eller debuggeren *gdb*, dette var litt vanskelig til å begynne med men etterhvert gikk det greit. Under funksjonalitets testingen ble det oppdaget flere feil, men disse ble greit rettet opp.

Måten dette ble gjort på var feks. å skrive ut stringer, variable, kommentere bort kritiske funksjoner ol. og se på disse om feilen kunne ligge der. *Strace* ble også brukt.

7.2 Testing av ferdig utviklet system

Når det hele var satt sammen til en hovedapplikasjon begynte en omfattende test av funksjonaliteten. Her gikk vi utfra at de grunnleggende delene som layout og lignende var ok.

Vi hadde nå et sett med bilder i 100x100 pixel størrelse hentet fra `www.orl.co.uk`, sort/hvit, og disse mente vi var greie og test med, så vi la inn en del bilder i databasen og prøvde ut de forskjellige applikasjonene som feks. insert bilde i database, browsing i database og gjenkjenn. Vi fant etterhvert små feil som ble fort rettet opp, disse feilene stod mest i feil i løkker, arrayer og lignende. Etter en tids testing fant vi og ut metoder som ville være mer fornuftig enn det vi først hadde planlagt. Dette gikk hovedsaklig på layout som feks. bedre måte og liste ut bilder, legge knapper ol.

7.3 Feilhåndtering ved bruk av applikasjon

For å lette brukers forståelse ved bruk av applikasjonen har vi implementert popup bokser som gir bruker feedback på operasjoner som ble utført. F.eks. ved innlegging av bilde og data om person.

7.4 Testing av algoritmer

Denne testingen vil i stor grad gå på testing av algoritmer, med og uten preprossesering, og finne treff prosenter ved gjenkjenning.

Vi hadde et sett med sort/hvit bilder fra Olivetti Research Lab (`www.orl.co.uk`), 100x100 pixel, som vi la inn i databasen. Dette settet bestod av 10 bilder

a 40 personer. Vi valgte å legge inn 5 bilder pr person for da å ha mulighet til å teste med bilder som ikke er identiske.

Personene fikk sitt eget navn og ble kommentert med bilde 1-5. Ettersom bildene var veldig like når det gjelder lys styrke og kontrast valgte vi og å konvertere et utvalg av bilder. Konverteringen gikk ut på å la et bilde av en person som ligger i databasen og et bilde av samme person som ikke ligger databasen bli lysere, mørkere, mer kontrast, mindre kontrast. Vi valgte ut 5 personer som vi skulle konvertere. Dette resulterte i et sett med 40 bilder.

Dette fant vi fornuftig for da kunne vi teste hvor mye lys styrken og kontrasten hadde å si på gjenkjenningen, og hvordan dette ville virke inn på de forskjellige algoritmene med og uten preprossesering.

Vi laget så to sett av bilder som vi ville teste med.

Første sett skulle være med et orginal bilde og fire konverterte bilder identiske med et i databasen bortsett fra lys/kontrast endring. Dette kaller vi **Test sett 1**.

Andre settet skulle bestå av et orginal bilde og fire konverterte bilder som ikke er identiske med noen bilder i databasen, dvs. personen ligger i databasen men ikke det eksakte testet bildet vi kjørere gjenkjenning på. Dette kaller vi **Test sett 2**.

Dette vil bli mest realistisk ettersom man aldri vil få identiske bilder når man setter seg foran et kamera.

Testen ble foretatt ved at vi forsøkte å gjenkjenne 5 forskjellige personer ut fra definisjonen av **Test sett 1 og 2.** Dette innebærer at hvis man gjenkjenner alle 5 oppnåes det treff prosent på 100.

TEST 1:
Tester algoritmen L2METRIC ,med og uten preprossesering.
Testing foregår med **Test sett1**.

Algoritme	**Pre prossesering**	**Bilde**	**Treff**
L2METRIC	Convert to own intensity plane & Histogram equalizing	orginal	100%
		+contrast	100%
		-contrast	100%
		+bright	100%
		-bright	100%
L2METRIC	Convert to own intesity plane	orginal	100%
		+contrast	100%
		-contrast	100%
		+bright	100%
		-bright	100%
L2METRIC	Histogram equlizing	orginal	100%
		+contrast	100%
		-contrast	40%
		+bright	60%
		-bright	60%
L2METRIC	None	orginal	100%
		+contrast	100%
		-contrast	40%
		+bright	20%
		-bright	0%

TEST2:
Tester algoritmen L2METRIC, med og uten preprossesering.
Testing foregår med **Test sett2**.

Algoritme	**Pre prossesering**	**Bilde**	**Treff**
L2METRIC	Convert to own intensity plane & Histogram equalizing	orginal	60%
		+contrast	60%
		-contrast	60%
		+bright	60%
		-bright	60%
L2METRIC	Convert to own intesity plane	orginal	80%
		+contrast	80%
		-contrast	80%
		+bright	80%
		-bright	80%
L2METRIC	Histogram equlizing	orginal	60%
		+contrast	40%
		-contrast	20%
		+bright	20%
		-bright	20%
L2METRIC	None	orginal	80%
		+contrast	80%
		-contrast	0%
		+bright	20%
		-bright	0%

TEST3:
Tester algoritmen COVARMETRIC ,med og uten preprossesering.
Testing foregår med **Test sett1**.

Algoritme	**Pre prossesering**	**Bilde**	**Treff**
COVAR-METRIC	Convert to own intensity plane & Histogram equalizing	orginal	100%
		+contrast	100%
		-contrast	100%
		+bright	100%
		-bright	100%
COVAR-METRIC	Convert to own intesity plane	orginal	100%
		+contrast	100%
		-contrast	100%
		+bright	100%
		-bright	100%
COVAR-METRIC	Histogram equlizing	orginal	100%
		+contrast	100%
		-contrast	100%
		+bright	80%
		-bright	100%
COVAR-METRIC	None	orginal	100%
		+contrast	100%
		-contrast	80%
		+bright	80%
		-bright	100%

TEST4:
Tester algoritmen COVARMETRIC ,med og uten preprossesering.
Testing foregår med **Test sett2**.

Algoritme	**Pre prossesering**	**Bilde**	**Treff**
COVAR-METRIC	Convert to own intensity plane & Histogram equalizing	orginal	60%
		+contrast	60%
		-contrast	60%
		+bright	60%
		-bright	60%
COVAR-METRIC	Convert to own intesity plane	orginal	80%
		+contrast	80%
		-contrast	80%
		+bright	80%
		-bright	80%
COVAR-METRIC	Histogram equlizing	orginal	60%
		+contrast	40%
		-contrast	20%
		+bright	20%
		-bright	60%
COVAR-METRIC	None	orginal	80%
		+contrast	80%
		-contrast	20%
		+bright	20%
		-bright	80%

7.5 Konklusjon av testing

Det som fremgikk av testingen var som forventet. For **L2Metric** preprosesseringen størst innvirkning. Spesielt for de konverterte bildene. Når det gjelder **CovarMetric** hadde forprosessering mindre tilnærmet ingen innvirkning på test resultatet. Det må sies at testen er litt lite omfattende til å trekke for bastante slutninger, men gir en pekepinn på gjenkjennings evnen til systemet og hvordan de forskjellige preprosessering rutiner innvirker.

8 Installasjon og bruk

Her skriver vi om den totale gangen i installasjonen og så skriver vi litt navigasjons struktur

8.1 Installasjon

Systemet vårt er i første rekke tenkt kjørt på en *Linux* plattform, det er det som ligger til grunn for innholdet på installasjons cd'en for systemet. Men det skal være fullt mulig å kjøre systemet på for eksempel en MS Windows plattform, men dette vil kreve at den enkelte bruker selv erverver det som trengs av software, det ligger altså ikke på installasjons cd'en. Under følger de aktuelle internett adressene hvor man kan laste ned det som trengs, samt hvilke versjoner av programvare vi har bygget systemet på. Det skal ikke være problem med nyere versjoner en det som er oppgitt under, eldre versjoner kan vi ikke garantere for.

Installasjon av vårt system krever at du har installert:

- *Mysql* versjon 3.22.32 eller nyere som er selve databasen i systemet. Denne program varen kan lastes ned på: www.mysql .com.
- *Mysql++* API vesjon 1.78 en som utgjør interfacet mellom *Mysql* og *C++*. Denne program varen kan lastes ned på: www.mysql .com.
- *Qt* biblioteket som utgjør rammeverket systemet er utviklet rundt. Denne program varen kan lastes ned fra:`www.trolltech.com`

Skal man kjøre systemet mot et kamera må man ha den nødvendige software som trengs for de enkelte kamera, det vil vi ikke gå inn på det her. Vi legger i første omgang opp til at systemet kjøres uten kamera. Men på installasjons cd'en som følger systemet vil det også ligge en versjon av systemet som støtter bruk av kamera.

Det som følger installasjons cd'en for systemet:

- MySQL-3_22_32-1_i386.rpm
- MySQL-devel-3_22_32-1_i386.rpm
- MySQL-client-3_22_32-1_i386.rpm
- MySQL-shared-3_22_32-1_i386.rpm
- MySQL++-1.7.8_1.i386.rpm
- Qt-2.2.2-tar.gz
- Biri2 kildekode for kjøring **uten kamera** må kopieres inn på harddisk under ønsket mappe, og kompileres opp vha. make kommandoen.

- Biri2 kildekode kjøring **med kamera** må kopieres inn på hardisk under ønsket mappe, og komileres opp vha. make kommandoen.
- README fil for Biri2 med rekkefølge for installering, tips og kjente problemer
- Filen database.txt som inneholder database tabell definisjon, altså sql utrykket for å opprette den.
- Mappen orl_faces som inneholder 400 ansiktsbilder på storrelsen 100*100 piksler. Som er tenkt brukt ved testing.
- Dokumentasjon av klidekode i html, pdf format og ps

8.2 Bruk

Systemets arkitektur er utformet slik at det lett skal kunne videre utvikles enten i form av at man bygger videre på vårt system eller ved at man plukker ut klasser og inkluderer disse i helt nye systemer. Systemet er godt dokumentert, men med tanke på videre utvikling bør man ha en hvis kjennskap til *C++,Qt og Mysql.* Vi håper noen fatter interesse for det vi har utviklet, systemet har nok størst potensiale som videre utviklings prosjekt. Det er lagt godt til rette for at det lett skal kunne implementers flere gjenkjennings algoritmer. Men selv om systemet kun foreligger i en beta versjon, er det fullt mulig å nytte gjøre seg det selv om bruksområdet er noe begrenset foreløpig. Layouten til systemet er utformet slik at en bruker med moderate datakunnskaper lett skal kunne ta i bruk systemet. Det er med andre ord lagt opp til at det skal være selvforklarende. Systemet har under de tester vi har utført vist seg stabilt, og vi håper eventuelle brukere blir fornøyd med produktet vårt.

9 Diskusjon

9.1 Prosessen

Prosjektet startet for vår del ved at vi kom sammen og dannet selve prosjekt gruppen som da gikk under navnet JRF (Jarle Robert Frank), noe som hang igjen i fra tidligere prosjekt arbeid ved Høgskolen i Gjøvik, senere endret til Biri2. Dette skjedde allerede i slutten av 2000, før de aktuelle hovedprosjektene for våren 2001 ble presentert for oss. Gruppe sammensetningen kom egentlig av seg selv, ettersom vi hadde fra tidligere har utført flere mindre prosjekter sammen. Vi ble i utgangspunktet oppfordret til å søke nye gruppe sammensetninger i forbindelse med hovedprosjektet, men vi hadde gode erfaringer mht. sammarbeid og resultat fra de tidligere prosjektene vi hadde utført sammen, og valgte altså derfor å fortsette samarbeidet.

Etter presentasjon av de ulike prosjektene, startet utvelgelses prosessen av oppgave. Biri2 prosjektet pekte seg tidlig ut blant gruppens medlemmer, og det ble opprettet kontakt med oppdragsgivere: Erik Hjelmås og Ivar Farup. Vi var så heldige å tildelt oppgaven og dermed var det hele igang. Det første som skulle utarbeides var en kort beskrivelse av oppgaven, samt gruppe sammensetningen, dette ble levert hovedprosjekt koordinator for avdeling data, Tom Røise. Videre frem mot selve prosjektstarten i januar 2001fikk vi forskjellige tips av veilder: Erik Hjelmås, angående hva vi burde jobbe med, litteraturstudier i første rekke.

I januar 2001 startet vi arbeidet med forprosjekt rapporten, denne ble utarbeidet etter de rettnings linjer som er gitt for hovedprosjekter ved Høgskolen Gjøvik.
Forprosjekt rapporten (med som vedlegg A) går mer i detalj inn på:

- Oppgavebeskrivelsen/avgrensing av oppgaven
- Prosjektorganisering: ansvarsforhold, roller og bemanning
- Planlegging, oppfølging og rapportering
- Risiko
- Kvalitetssikring
- Fremdriftsplan, aktiviteter,milepæler og arbeidsmetoder

Under arbeidet med denne for-prosjekt rapporten brukte vi ulike metoder for å planlegge og estimere tidsbruk, organisering og kompleksitet. *Gannt*-skjema(følger med i vedlegg A) vart et viktig verktøy for å koordinere tids bruken noe som er viktig i et prosjekt.

Første del gikk med på planleggingen, men vi bestemte også tidlig rutiner når det gjelder versjons-kontroll, backup-rutiner, skriving av referater, status rapporter.

Nå i ettertid er det interessant å se hvordan vi har lykkes med det som ble definert i forprosjektrapporten, som har fungert som et styringsverktøy gjennom prosessen. Vurderinger i forhold til for-prosjektet:

- Vi undervurderte vel kanskje kompleksiteten i å programmere i et *UNIX/Linux* miljø. Vi hadde tidligere kun erfaring i *C++*-programmering i et *MS Windows* miljø vha. Borland C++ builder. Slike verktøy ordner alt av kompilering og linking selv og vi hadde ikke erfaring med grafisk programmering eller bruk av spesielle bibliotek under *C++* før dette prosjektet. Vi måtte her lære oss grunnleggende bruk av verktøy som *Emacs, CVS, tmake, kompilering, linking, strace, gdb, oppsett av Makefile* osv.

- Generelt kan vi vel også si at prosjektet satte høyere krav til programmeringskunnskap og foståelse av *C++* en det vi hadde før prosjektets begynnelse. Dette førte til at en del av koden vi brukte i starten ble omskrevet.

- Oppgavebeskrivelsen/avgrensingen av oppgaven som vi definerte i forprosjekt rapporten kan vi konkludere med at vi i all hovedsak har klart å følge den, men det ble noen avvik. Vi så etter hvert at nytten av at vår del, av det helhetlige *BIRI* systemet skulle kunne kjøres uten *BIRI1* og *BIRI3* sine moduler, som en frittstående applikasjon og utviklet derfor et ekstra system som kan kjøres helt uavhengig. Dette innebar ikke mye ekstra arbeid for vår del, da det meste som skulle til mhp. implementasjon allerede var på plass.

- Det litt spesielle valget vi gjorde ved ansvarsforhold i prosjektet ved at prosjektleder ansvaret skulle gå på omgang slik at alle fikk prøvd rollen, har vært ganske interessant. Her fikk alle prøvd seg. og selv om vi kanskje har litt forskjellige tanker om hvordan man skal organisere delegering og samordning av aktivitetene, synes vi at vi har unngått de store konflikter og vi har ikke opplevd for store spørsmål om den videre framdriften i prosjektet. Vi innser at denne ansvars fordelingen kun vil fungere i lærings sammenheng for mindre prosjektgrupper som vår er.

- Planlegging, oppfølging og rapportering. Når det gjelder planleggingen har i det daglige arbeidet først og fremst jobbet opp mot de enkelte milepælene og lagt opp aktiviteten etter de. Oppfølging av de enkelte beslutninger og problemer som er dukket opp underveis har vært basert på at de enkelte gruppemedlemmer har blitt tildelt ansvar og gitt tidsfrister. Dette var også slik det var tiltenkt i forprosjektrapporten, det vi derimot har vært litt for liberale med er skriving av møterapporter. Vi har ikke helt klart å etterleve de krav vi satte oss. Statusrapportering har fungert bra, både i form av god muntlig kontakt med veileder og skriftlige rapporter. De fire statusrapportene som er utarbeidet finnes som vedlegg til hovedprosjektrapporten.

- Av risiko elementene vi identifiserte i forprosjektrapporten, har noen slått til. De har kommet litt uventet, men vi synes vi har løst disse problemene på en grei måte. Vi hadde en ukes fravær på en av gruppe medlemmene grunnet sykdom, men vedkommende var sitt ansvar bevist og sørget for å holde seg oppdatert og utføre de oppgaver som var tillagt ham (takk til

Frank der). Et stort risiko moment var teknologiske problemene som oppstod da vi valgte implementasjons verktøy. Det oppstod komplikasjoner i grensesnittet mellom de ulike typer verktøy vi bruket under utviklingen. Et problemet oppstod under installasjon av *Mysql++ API*. *C*-kompilatoren versjon gcc-2.96 som fulgte Redhat Linux 7.0 som vi kjørte på utviklings maskina. *Mysql++ API'en* lot seg ikke installere, men etter vi kjørte oppdatering på *C*-kompilatoren ble problemet løst. I hele tatt synes vi at *MySQL++* api var begrenset dokumentert, men vi fant det utfordrende å grave opp mer informasjon om dette nye programmerings interfacet. Vår del av systemet var i utgangspunktet veldig avhengig av de andre *BIRI*-gruppenes produkt, særlig *BIRI1* som jobbet med ansikts deteksjon. For å eliminere usikkerheten valgte vi å lage en selvstendig versjon av vår del av systemet. Vi fikk integrert mot *BIRI1*, men dessverre ikke mot *BIRI3* da de ikke ble ferdig i tide. Risikoen for tap av data og versjons kontroll, løste vi vha bruk av *CVS*. Alt arbeid lå hele tiden i minst to eksemplarer på lokal utviklings maskin og på *CVS* server, og vi kom aldri i vanskeligheter pga tap av data.

- Kvalitetssikring. vi hadde i forprosjektrapporten plukket ut de punkter vi fant nødvendig ut fra *IEEE-730* standarden. Disse punktene har vi fulgt, med unntak av et som går på kode standarden. Den skulle i utgangspunktet ha fulgt *GNU* standarden, men vi fant etter hvert ut at dette ble litt omfattende for vårt bruk og tok med det vi fant fornuftig. Ellers bør det nevnes at vi har utført flere tester på systemet både mhp. gjenkjenning av ansikter og håndtering av feilsituasjoner. Resultatet av testen på ansikts gjenkjenning finnes i kapitlet om testing i hovedrapporten. Håndtering av feilsituasjoner er i form av forskjellige sjekker og via bruker-meldinger.

- Vurderinger i mht. fremdriftsplan i forhold til aktiviteter, milpæler og arbeidsmetoder Vi har i all hovedsak fulgt de aktiviteter som fremgikk av den opprinnelige fremdriftsplan. Milepælene som ble satt opp ble også overholdt, med unntak av den 4. som gikk en uke over tiden og den tredje som ble framskyndet 2 uker pga at vi drev med et annet prosjekt i faget Klient og server sideprogrammering ved Høgskolen. Detaljene rundt aktivitetene og milepælene kan ses på i statusrapportene, som ligger som vedlegg D. Vi valgte *RUP* som arbeidsmetode/utviklingsmodell (jfr kap.1). Erfaringene fra dette må sies å være interessante. Fra tidligere hadde vi ingen praktisk erfaring med modellen, men den viste seg å være effektiv til vårt prosjekt. For oss innebar prosjektet bruk av for oss ukjente algoritmer og utviklingsverktøy, og for at det hele ikke skulle bli uoversiktlig og for komplekst, passet det bra med en modell som var inkrementell (noe utvikling-paradigmen av *Open Source* programvare ofte er). *RUP* hjalp oss også å identifisere og håndtere risiko både tidlig og lengre ut i prosjektforløpet vha flere iterasjoner og fortløpende risiko-vurderinger. Selv om vi ikke fulgte modellen slavisk, noe som etter vår mening heller ikke er nødvendig, synes vi har opparbeidet oss god erfaring med metoden. Alle

utviklingsprosjekter er forskjellige og *RUP* er åpen for egne tilpasninger. Konklusjonen blir at *RUP* fungerte godt til vårt prosjekt.

- Utviklingen av algoritmer i prosjektet var litt uviss, fordi når vi startet prosjektet trodde vi i utgangspunkt at vi skulle bruke noen ferdig konstruerte algoritmer som oppdragsgiver hadde. Vi følte at for at vi skulle få et system som vi skulle være fornøyde med, ville vi ha på plass noen algoritmer som fungerte noenlunde tilfredsstillende. Vi fant ut at de algoritmene som vi trodde vi skulle få fra oppdragsgiver ikke kom på plass og begynte selv å undersøke mulighetene for mer kompliserte algoritmer enn den han foreslo (en algoritmen som ble foreslått implementerte vi i en test applikasjon tidlig i prosjektet for å teste verdien av). En av prosjektdeltagerne ble satt på jobben. Han satte seg ned og undersøkte og gjorde en del studier i bøker og publikasjoner rundt temaet. Etter en del av disse undersøkelsene, følte vi at vi hadde litt bedre oversikt i hva som var vanlig i slike gjenkjennings systemer og hadde gjort en vurdering av kompleksitets-graden av dqe forskjellige metoden og vi bestemte og for hva vi mente var tilfredsstillende for oss å implementere i et slikt prosjekt som vi har.

I og med at en del av prosjekt deltakerne ikke hadde tilgang til nettverket på skolen fra sine hjem, brukte vi et selv konstruert fildelings system implementert i *Java*, *FileShare1*. Dette fordi vi ville slippe og hele tiden gå rundt med disketter og slike medier da det ble veldig uoversiktlig og og vanskelig å holde rede på versjoner. Dette *FileShare1* web programmet fungerer via en browser og jobber mot en *MySQL* database. Uten å skryte så vil vi si at dette har gjort mulighetene for prosjektmedlemmene å jobbe hjemmefra på en sikker og oppdatert måte, mye større. *FileShare1* finnes som eksempel på `http://cgiweb.hig.no/rob_stra/FileShare1`.

9.2 Hva har vi oppnådd?

Vi har oppnådd de største målene vi satte oss for dette prosjektet. Vi har klart å implementere et relativt velfungerende system som kan ha et bra anvendelses område og som en bra start på videreutvikling av systemet.

Vi har klart å integrere oss mot *BIRI1* modulen som vi hadde planlagt, noe som gjør at vi kan ta imot bilder fra et web-kamera og kjøre en gjenkjenning av personen det blir tatt bilde av, mot databasen. Integrering mot *BIRI3* modulen har vi ikke løst ved innleverings tidspunktet. Vi prøvde å skrive et enkelt interface mot denne modulen, men vi fikk ikke testet dette fordi *BIRI3*-modulen ikke vart ferdig i tide. Ved siden å ha en felles applikasjon sammen med *BIRI1*-modulen, lagde vi en egen applikasjon som kunne fungere uavhengig av de andre modulene og fungerer som et slags foto-bok system med opplasting av bilder fra disk.

Vi har klart å installere og satt opp en database der man klan lagre bilder i databasen. Vi har klart å finne fram til og bruke verktøy som fungerte sammen

og som var ukjente for prosjekt-deltakerne på forhånd og som var definert av oppdragsgiver på forhånd uten at han viste at dette faktisk kunne løses ved hjelp av disse.

Vi har implementert to forskjellige algoritmer for gjenkjennings prosessen og to forskjellige pre-prosesserings metoder. Dette for å undersøke hvordan de ulike algoritmene fungere og viktighetsgraden av pre-prosessering av et bilde før gjenkjenning.

9.3 Drøfting av løsningen

Generelle løsninger er diskutert underveis i rapporten. Her drøfter vi mere hva som vi mener er oppfylt i forhold til kravspesifikasjonen.

Vi synes valget av database har vist seg fornuftig, fordi *MySQL* i seg selv er enkelt oppbygd og håndterer data på en meget effektiv måte. Samtidig er den plattform uavhengig som gir løsningen vår større fleksibilitet.

Arkitekturen i løsningen var viktig for oss, for å følge three-tier architecturen [6]. Dette innebærer at laget under ikke har direkte tilknytning til overliggende lag. Dette synes vi har klart i koden da vi har skilt ut database håndteringen i egne klasser. Selve forretnings-logikken i applikasjonen, middle-layer, har vi skilt ut og det er her selve løsningen med henhold til gjenkjenning ligger og håndtering og rangering av kandidater for gjenkjenning. Presentasjonslaget er skilt ut på en enkel måte og tar hånd om all layout. Dette gjør at endringer i layouten gjøres i disse klassen og ikke må gjøres gjennom hele applikasjonen.

Navigasjonsstrukturen er vi fornøyd med fordi den er så selvforklarende og enkel, slik at selv den uerfarne bruker lett skal kunne benytte seg av applikasjonen. Vi mener også vi har klart å fange opp forskjellige feil-situasjoner som kan oppstå i bruker interaksjoner ved å sjekke input og skrive ut rettledninger og feil-meldinger.

Det som mangler i systemet er når man står i gjenkjennings skjerm bildet og ønsker å fange et nytt bilde fra kamera for gjenkjennnig uten å gå helt ut av dette vinduet. Dette vart ikke løst fordi vi ikke fant ut noen god metode for å fange et nytt bilde fra kameraet ved hjelp av en interaksjon i dette gjenkjennings vinduet. Dette problemet gjelder også for den delen der man skal legge bilde inn i databasen. Man kan ikke ta et nytt utklipp uten å gå veien om hoved vinduet.

Algoritmene som ble implementert er ikke av de mest kompliserte. Vi tok i bruk algoritmer som fungerer brukbart når man har gode billedfangster å ta utgangspunkt i. **L2Metric** regner ut den *Euclidske* avstanden mellom to bilder. Denne algoritmen tar ikke noe hensyn til samvariasjon mellom to bilder. Man sammenligner kun pixel for pixel og denne algoritmen er svak når det endrede lys og kontrast forhold. Den andre algoritmen vi implementerte baserte seg på korelasjon mellom to bilder, **CovarMetric**. Denne tar mere høyde for samvariasjon mellom to bilder og skal være mere tolerang for endrede lysforhold og kontraster i bildet. Sammen med disse algoritmene laget vi noen preprosesserings rutiner som ga bedre forhold med bilder som hadde endrede lys -og kontrast forhold. Selv om disse algoritmene fungerer rimelig bra i vår implementasjon,

så finnes det mer brukte og sofistifikerte metoder som antakelig løser gjenkjenningen av personer som ikke det er så gode og presise bilde-fangster av, på en bedre på (jfr kap.3). Men ofte er disse mer regne og prosesserings krevende noe som vil gjøre applikasjonen langsommere. Vi hadde heller ikke som mål i denne oppgaven å implementere "state-of-art" gjenkjennings metoder, men heller lage en applikasjon der man enkelt kunne legge til nye og bedre algoritmer. Hvordan de ulike metoden fungerte testet vi i kap.7 på et datasett vi hentet fra Olivetti Research Laboratory. Testene ga et brukbart resultat og vi er ganske fornøyd med at de algoritmene vi implementerte faktisk fungerte så bra som de gjorde.

Integreringen med *BIRI1*-prosjektet gikk også rimelig bra, og det mener vi var ut ifra at vi var tidlig i utviklings prosessen opptatt av å lage en grei interface mellom de forskjellige modulene. Løsningen der *BIRI1* og *BIRI2(vårt prosjekt)* er integrert fungerer på etter vårt syn tilfredsstillende måte. Integreringen mot *BIRI3* -modulen fikk vi ikke i orden og i forhold til den totale løsningen som det var krav, vart dette ikke løst. Dette var utenfor vår kontroll og vi mener vi gjorde alt klart for en integrering, men en løsning fra *BIRI3*-prosjekt gruppen kom ikke i tide.

9.4 Hva har vi lært?

Vi har fått prøvd kunnskapene vi har tilegnet oss i løpet av 3 år på Høgskolen i Gjøvik i praksis gjennom prosjektarbeidet. Tekniske utfordringer har dominert, men vi har også fått brynt oss på mer systemutviklings spørsmål når vi har jobbet med prosjektet.

Vi bestemte oss tidlig i prosjekt perioden for å benytte en system utviklings modell som er utbredt i arbeidslivet og i større utviklings prosjekter. Dette gjorde vi for vi ville prøve å få erfaringer i bruken av disse rammeverkene for senere bruk i jobb-sammenheng. Dette synes vi alle i prosjekt gruppen har vært en lærerik prosess der vi har klart å fylle litt "kjøtt" på de "teori-benene" vi har fra før. Selv om bruken av disse rammeverkene kan virke ganske uforståelige i et slikt begrenset prosjekt som disse student-prosjektene er og av denne karakter, så har vi tvinget oss i gjennom dette med ønske om mer læring for øye.

Vi har lært oss mer inngående å bruke *Linux* som operativ system og gjort nye erfaringer. Bruken av nye og for oss ukjente verktøy har stått i fokus hele tiden.

Vi har fått mer praktisk erfaring i mer langvarige prosjekter enn det vi hadde fra før i utdannelsen ved Høgskolen i Gjøvik. Dette har gjort oss mer beviste på arbeidsmetoder og vurdering av forhold som spiller inn i en prosjektgruppe. Samt at vi, i og med at det helhetlige *BIRI*-prosjektet var delt inn i 3 moduler/grupper, gjort erfaringer i hvordan prosjekt gruppene kan ha ulike incentiver i det felles prosjektet.

Her er en summarisk oversikt over hva vi har lært:

- *CVS*

System som tar seg av de forskjellige versjonene av filene våre og håndterer disse på en svært god og effektiv måte. Dette gjør at vi, om vi ønsker, kan gå tilbake til hvilken som helst versjon og hente ut filene slik de var på det tidspunkt. Når vi legger inn nye versjoner legger vi på en liten kommentar som forteller om endringen i forhold til forrige versjon. på denne måten blir det veldig oversiktlig å se hva som er gjort når.

- *Emacs*

Programmet som er en tekst editor virket i utgangspunktet som tungvint og lite hensiktsmessig å bruke til koding. Etter hvert som vi lærte oss å bruke programmet har denne oppfatningen endret seg. Vi har etter å ha blitt kjent med programmet,som så mange ander før oss at det er svært effektivt og godt å jobbe med.

- C++

Mer inngående kunnskap i C++. Vi føler vel kanskje at vi kanskje står med like mange spørsmål som svar etter å brukt programmerings språket som er komplekst og vi føler at det er mye mer som vi kan lære i bruken av det.

- *Qt*

Vi har lært og bruke Qt på en litt overfladisk måte. Vi har bare skrapt litt på overflaten av hva Qt kan brukes til. Vi sitter iallfall igjen med et inntrykk av et meget godt utviklet utviklingsprogram for kryssplattform prosjekter.

- *gcc*

Vi hadde i faget "datamaskin organisering" ved Høgskolen i Gjøvik lært en del av gangen i programmering i prosessen i kompileringen og linking, men de fleste av oss var vel ganske usikre da vi tok fatt på programmeringen under *Linux* ved hjelp av *gcc*-kompilatoren. Vi har lært en hel del omkring det å kompilere kode og linke objekt filer sammen som vi mener har gitt oss en mere dyp-gående erfaring enn det å sitte å programmere vha. en programmerings-workshop som f.eks. Borland C++ builder.

9.5 Evaluering

9.5.1 Gruppas arbeid

Når vi startet på selve oppgaven i januar 2001var det en del frem og tilbake. Det første vi måtte gjøre var å få rigget oss til på gruppe rommet vi var blitt tildelt. Dette rommet var når vi startet rensket for teknisk utstyr og møbler, det ble noen turer innom vaktmesterene på jakt etter utrangerte møbler som vi kunne låne. Gruppen gikk til innkjøp av en egen pc, den ble kjøpt i deler og satt sammen. Videre installerte nødvendig programvare til bruk i prosjektetet. Uten grupperom og investering i egen pc hadde nok arbeidet med prosjektetet

vanskelig latt seg gjøre, vi har hatt stor nytte av å kunne bruke skolens nettverk og at alle *BIRI* gruppene har vært samlet på samme sted har lettet jobben med koordinering mellom gruppene. Det hersket litt uklarhet helt i starten rundt selve oppgaven og hvor vi skulle begynne, men etter noen oppklaringsrunder bestemte vi oss for hvordan vi skulle angripe oppgaven. Samarbeidet i gruppen har fungert bra. Selv om prosjekt medlemmene satt med forskjelllige kunnskap og interesser, klarte vi å få ting gjort. Vi hadde på forhånd satt opp når det skulle jobbes med prosjektet. Den tiden vi hadde satt av var ikke nok, så det ble mye jobbing ut over de oppsatte timene/dagene . Grunnen til at vi klarte å følge fremdriftsplanen så godt som vi har gjort, har sin forklaring i at vi har gjort det som skal til for å holde milepælene. Dette innebar jobbing på kvelder, helger og iblant natt. Alle gruppe medlemmene har ved siden av skolen andre ting(jobb, hobbyer, kjærester osv), så det har vært ganske hektisk til tider. Særlig i forbindelse med eksamener og innleveringer. Vi er alle enige om at det skal bli godt å puste ut noen dager etter et hektisk siste halvår.

9.5.2 Veilederen

"Rollen til veileder er å følge opp og påse at høgskolen interesser i kontraktssammenheng blir ivaretatt, gi råd til studentene i forbindelse med gjennomføringen av prosjektet"-ref. Retningslinjer for hovedprosjekt ved HIG.

Vi hadde satt i forprosjekt rapporten at vi skulle ha formelle møte med veileder hver 3. uke. Vi gjennomførte 2 formelle møter, men siden oppdragsgiver i prosjektet er ansatt ved Høgskolen i Gjøvik og at vi var i kontakt med veileder en del utenom disse formelle møtene, gikk vi litt i bort fra dette. Vi hadde diskusjoner med veileder under hele prosessen noe som vi synes har fungert bra.

Veilederen i vårt prosjekt er også oppdragsgiver, noe som kan gi et litt motsettnings forhold i og med at veileder kanskje skal være en mer "dempende" karakter, mens oppdragsgiver kanskje har den formen at han tar å drar prosjektet i alle kanter. Dette er vi studentene i prosjektet klar over og rollen til veileder/oppdragsgiver i ulike sammenhenger måtte tolkes av oss (hva snakker han SOM nå osv.). Vi har ikke noe å utsette på veilederen i så måte og vi vil understreke at vi opplevde han som en meget positiv person i vår prosjekt gruppe med som interesse for prosjektet og med stor vilje til å stille opp både sent og tidlig.

En ting som vi kanskje synes veileder burde gjort, er å utdype at "nå snakker jeg som min rolle som veileder" (jfr over). Vi opplevde disse rollene litt frustrerende i starten av prosjektet. Vi følte det at prosjektet var dratt i alle forskjellige retninger når det gjaldt hva som skulle være hoved fokusen i oppgaven. Men etterhvert følte vi at alle fant ut hvordan vi skulle gå fram, og samarbeidet har gitt oss mye og veileder har vært en stimulerende kraft for oss.

Vi i prosjekt gruppen forlangte aldri at veileder skulle være "guru" innenfor den utviklings teknologien vi skulle bruke, men at han hadde litt oversikt og interesse. Dette mener vi at vi har opplevd med veileder da han har vist stor interesse for teknologiske løsninger og kommet med både spørsmål og svar som har ledes oss i riktig retning.

9.5.3 Oppdragsgiveren

Oppdragsgiver var den samme som veileder. Disse rollene har som forklart i punktet over, vært litt forvirrende. Når vi først fant ut hvilken Erik Hjelmås spilte (da han var både oppdragsgiver og veileder), opplevde vi han som vi mener en oppdragsgiver ofte gjør. En typisk karakteristikk mener vi at denne personen er å drar prosjektet i forskjellig retninger og kanskje kommer med nye eller endrede krav ettersom prosjektet skrider fram. Da blir det opp til prosjektmedlemmene og veileder og vurdere dette.

Vi skulle ha formelle møter som forklart i punktet over hver 3. uke, men dette gikk vi mer en mindre bort i fra pga. oppdragsgiver/veileder og prosjektgruppens lokalisering og kontakt.

Vi mener at oppdragsgiver har vært en motiverende person som har gjort oss mer nysgjerrig på fagområdet. En ting vi kan utsette på oppdragsgiver, er hva som ble forklart av hva som var av ferdig lagde algoritmer og metoder for selve gjenkjennings proseseringen. Vi mener vi ble forespeilet at disse algoritmene skulle ligge klare med god dokumentering og forklaring slik at vi ikke trengte å grave oss ,for dypt ned i de omfattende teoriene innen dette. Men heller konsentrert oss om å få de anvendt. Her opplevde vi at vi måtte delegere en person til å undersøke dette området og vurdere hvilken metoder som egnet seg, noe som vi ikke så for oss når prosjektet startet.

Men alt i alt er dette ting som vi som prosjekt gruppe kanskje burde kartlagt bedre, men samtidig er det vanskelig å anslå kompleksiteten når man ikke kjenner teorien bak det.

9.6 Oppsummering

Prosjektet har gitt oss mye ny kunnskap, både spesifikt om fagområder som programmering, operativsystemer, generelt om prosjektarbeid, systemutvikling, versjonskontroll og spesielt egne begrensninger og kunnskaper. Vi har fått en en viss forståelse for hva det innebærer og jobbe flere prosjektgrupper mot et felles mål og hvordan man står i et veileder/oppdragsgiver vs. prosjektgruppe forhold.

10 Underskrift

Gjøvik 22.05.01

Jarle Rudihagen Robert Strand Frank Svanheld

11 Litteraturliste

Referanser

[1] *Elementary Linear Algebra.* Pretice Hall Inc, 1988.

[2] *Object Oriented Programming in C++.* Wait Group Press, secound edition, 1995.

[3] *Prosjektarbeid.* Universitetsforlaget, 1995.

[4] *Software Engineering.* Addison Wesley Publishers Ltd, 1995.

[5] *Image Processing.* Pretice Hall PTR, 1997.

[6] *Applying Uml And Patterns.* Prentice Hall PTR, 1998.

[7] *Innføring i Sannsynlighetsregning og Statistikk.* Cappelen Akademiske Forlag, 1998.

[8] *MySQL.* New Riders, 2000.

[9] *Programming with Qt.* O'Reilly, 2000.

[10] *The Rational Unified Process an Intoduction*, chapter four. Addison Wesley Longman Inc., secound edition, 2000.

[11] *Teach Yourself Qt Programming in 24 Hours.* SAMS, 2000.

[12] *Qt Programming.* Prentice Hall PTR, 2001.

[13] A.Pentland, R.W.Picard, and S. Sclaroff. Photobook: Content-based manipulation of image databases. *Massachusetts Institue of Technology*, 1995.

[14] Rama Chellappa, Charles L. Wilson, and Saad Sirohey. Human and machine recognition of faces: A survey. 1995.

[15] Jau-Yen Chen, Charles A. Bouman, and John C. Dalton. Similarity pyramids for browsing and organization of large image databases. *Ricoh California Research Center.*

[16] Anil K. Jain, Robert P.W.Duin, and Jianchang Mao. Statistical pattern recognition: A review. 2000.

[17] Ivar Farup og Erik Hjelmås. Experimental comparison of face/non-face classifiers. *Hig*, 2001.

12 Vedlegg

12.1 Innholdsfortegnelse vedlegg

- A. Forprosjektrapport
- B. Fremdriftplan
- C. Kravspesifikasjon
- D. UML modellering
- E. Statusrapporter
- F. Dokumentasjon av kode

[6, 2, 1, 17, 5, 8, 11, 9, 12, 10, 7, 3, 4, 15, 16, 14, 13, 3]