11/21/2012

Gruppe 2

Informatikk Prosjekt 1 – IT1901

Alf Håkon Lille-Mehlum
Erik Frøseth
Ingeborg Ødegård Oftedal
Mathias Tempelhaug
Tord Kloster
Tuyet Trang Thi Pham

21 November 2012

Norges teknisk-naturvitenskapelige universitet

INNHOLD

1 INTRODUKSJON

1.1 PROSJEKT

Denne rapporten er et resultat av prosjektet som ble gjennomført i faget IT1901 - Informatikk prosjektarbeid I, ved Norges Teknisk-Naturvitenskapelige Universitet i Trondheim høst 2012. Prosjektet er gjort av 6 studenter. Medlemmene i denne gruppen er fra linjene Bachelor-informatikk og Bachelor-musikkteknologi.

1.2 BAKGRUNN FOR PROSJEKTET

Prosjektet har som bakgrunn å sette fokus på de forskjellige arbeidsoppgavene som vil kunne oppstå under utviklingen av virkelige datasystemer. Selve programmeringen vil foregå i Java for å utvikle gruppemedlemmenes kjennskap til dette programmeringsspråket. Vi skal også spesifikt sette oss inn i det grafiske brukergrensesnittet og dokumentasjon av kode. Andre rent praktiske mål, vil være å innføre kunnskap om hvordan databaser kan benyttes i utviklingen av større datasystemer. Det vil også være nødvendig å velge et versjonskontrollsystem, og sette opp det aktuelle slik at det kan benyttes for deling av kode. Medlemmene skal lære prinsippene bak

prosjekt-styringsrammeverket Scrum, og benytte seg av dette for å kontrollere utviklingen av prosjektet.

1.3 Om faget

Faget IT1901 går ut på at grupper med studenter skal gjennomføre et programmeringsprosjekt, som skal forbedre kunnskapsnivået i javaprogrammering og gruppearbeid. Hovedmålet er å gi studentene erfaringer med helheten i systemutviklingsprosessen. Dette krever forståelse i sammenhengen mellom produkt og prosess - relaterte problemer i et programmeringsprosjekt. Vi skal lære metoder og teorier som er vanlig å bruke i slike prosjekter.

1.4 Problemstilling

Vi jobber som programmerere i et programvareselskap som blir kontaktet av en bonde (sauebonde) som ønsker hjelp til å utvikle et system for å registrere og administrere sauene han forvalter. Vi har fått i oppdrag å komme med forslag til hvordan han/hun vil ha det, og å utvikle et slikt system. Det er flere forskjellige kriterier vi må forholde oss til, blant annet hvordan serveren og databasen må reagere på ulike meldinger fra sauene. Hovedkravet i dette prosjektet er at vi skal lage et program der man henter informasjon om hver enkelt sau fra en database. Hvis en sau blir drept eller skadet skal det sendes en alarm fra sauen til systemet, og derfra til den aktuelle sauebonden. Resten av kriteriene står nærmere forklart i kravspesifikasjonene.

2 Prosjektorganisering

2.1 Arbeidsprosess

Vi har felles arbeidsøkter to ganger i uken hvor vi har sittet sammen og jobbet.

- Mandag/Tirsdag kl. 14-18.
- Torsdag kl. 12-16.

Dette har gitt oss en fin mulighet til å diskutere løsninger på problemer som har dukket opp etter forrige økt, og hvordan vi har lyst til å gå videre. Etter hver arbeidsøkt har vi fordelt oppgaver som skal være gjort til neste gang, så vi har også jobbet mye på egenhånd. Merk at det står mandag/tirsdag, da valget av hvilken dag har blitt tilpasset studentenes timeplan.

Arbeidsoppgavene til den enkelte har blitt gitt i forhold til ansvarsfordelingen vi har hatt *(mer om dette under ansvarsområder 2.2 og gruppens kunnskapsnivå 2.3).* Vi har hatt et godt samarbeid på gruppa, noe som har gjort at produktet vi har laget har blitt bedre. Vi har hatt Scrum møter hvor medlemmene har diskutert hva vi har fått gjort, eventuelt ikke fått gjort. *(Les mer om Scrum se 3.3.)*

Dette har gjort at hele gruppa har veldig god innsikt i hvordan vårt produkt fungerer, og har gjort samarbeidet lettere.

Vi begynte prosjektet med å legge opp en plan for hvordan vi skulle løse de forskjellige problemene, og hva som var en god ide å starte med. Det vi i hovedsak begynte med var å lære oss Scrum, og organisere hvordan vi ville utføre prosjektet. Selve jobbingen med prosjektet handlet i starten mye om brukergrensesnitt (GUI), og hvordan den skulle se ut. I et av de første møtene, lagde vi store skisser på tavle for å tidlig ha et konkret forslag på hvordan grensesnittet skulle se ut *(se vedlegg "møtereferat" for bilder av skisse).* Vi lagde senere blant annet prototype og testet dette. Videre begynte vi med selve kravspesifikasjon. Deretter har vi jobbet mye vekselsvis med GUI og programmeringen. Parallelt med dette har vi jobbet med denne rapporten. Vi har lagt ved et vedlegg som beskriver problemer og løsninger til det vi ikke har fått tid til å ordne. *(Vedlegg, videreutvikling)*

2.2 Ansvarsområder

Alle på gruppa har fått ansvaret for egne områder i prosjektet, og dette har fungert veldig bra. I begynnelsen tok vi en runde over hvem som var flinke til hva, og hvem som hadde lyst til å gjøre hva. Det har vært viktig for oss at en skal få kunne jobbe med det en vil, og det en føler en mestrer. Det er derfor har ansvarsområdene blitt fordelt slik. At det blir fordelt på den måten er også vanlig innenfor Scrum-metodikken. I starten praktiserte vi ikke dette på like stor måte som på slutten av prosjektet, da de ulike rollene i gruppa ble tydeligere etterhvert. Selv om vi har fordelt ansvarsområder, har vi også samarbeidet og hjulpet hverandre. Arbeidsmengden til de forskjellige i gruppa har også vært veldig lik.

Alf Håkon Lille-Mæhlum:

- Graphic User Interface (GUI)
- Rapport
- Programmere

Erik Frøseth:

- Database og simulator
- SVN-admin
- Programmere

Ingeborg Ødegård Oftedal:

- Rapport
- Kravspesifikasjon
- Testing

Mathias Tempelhaug:

- Programmere
- Sprinter

Tord Sørenes Kloster:

- Programmere
- Rapport
- Testing

Tuyet Trang Thi Pham:

- Graphics User Interface (GUI)
- Rapport
- Kravspesifikasjon

2.3 GRUPPENS KUNNSKAPSOMRÅDER

Gruppen består av studenter som forventet har en god variasjon av kunnskaper. Dette gjør at vi har nok dybde til å komme gjennom prosjektkravene. Vi har medlemmer fra linjen Bachelor i Informatikk, der Tord, Ingeborg, Alf Håkon, Tuyet og Mathias er representert. Vi er også så heldige at vi har med oss Erik fra Musikkteknologi.

Alf Håkon Lille-Mæhlum:

- Generell programmering
- Systemutvikling erfaring

Erik Frøseth:

- Databaser
- Generell programmering, spesielt innenfor C#
- Erfaring med UML

Ingeborg Ødegård Oftedal:

- Generell programmering
- Litt erfaring om UML
- En del om databaser

Mathias Tempelhaug:

- Generell programmering
- En del om databaser

Tord Sørenes Kloster:

- Generell programmering
- En del om databaser

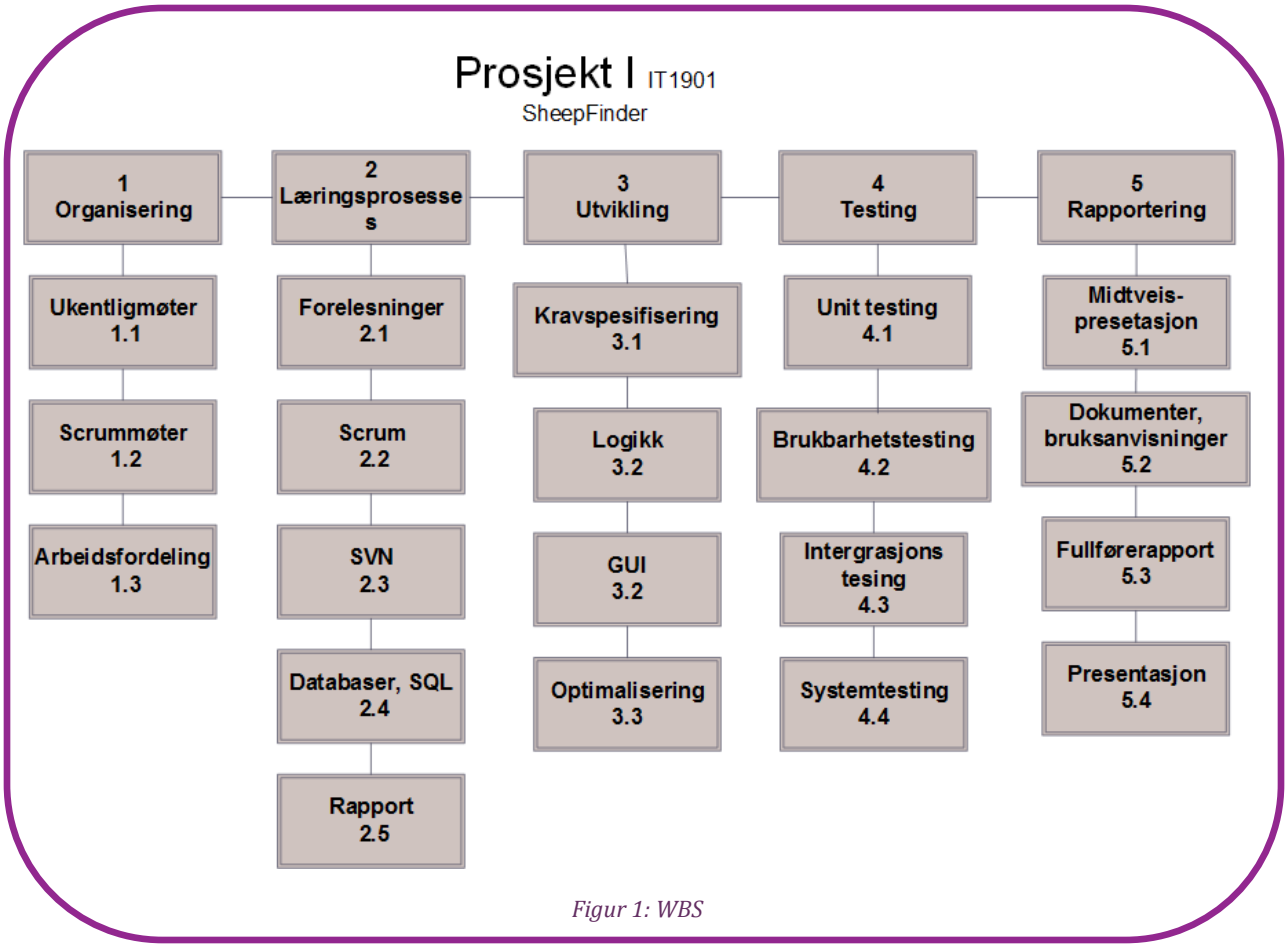
Tuyet Trang Thi Pham:

- Generell programmering
- Generell brukergrensesnitt
- En del om UML
- En del om Scrum

3 UTVIKLINGSPROSESS

3.1 WORK BREAKDOWN STRUCTURE

Arbeidsnedbrytingsstruktur, brukes til å organisere prosjektets oppgaver for å på en enkel måte vise en oversikt over hva som skal gjøres. Ved å organisere oppgavene på en slik måte, vil det bli enklere og estimere hvor mye tid man har til rådighet.



Figur 1: WBS

3.2 SCRUM

Scrum er et rammeverk brukt til å utvikle nokså komplekse datasystemer. Denne systemuviklingsmodellen faller under kategorien agile systemutviklingsmodeller (på norsk ofte kalt smidige utviklingsmodeller). Dette vil si at Scrum dermed er en systemutviklingsmodell som er basert på iterativ og inkrementell utvikling. Iterativ utvikling vil si at et handling blir gjentatt flere ganger. I Scrum vil dette bety blant annet man jobber i såkalte sprinter. Inkrementell

utvikling betyr at man jobber med ett inkrement om gangen, før man kan gå over på det neste. Dette gir også iblant fordelen at deler av systemet som utvikles kan leveres og tas i bruk før hele systemet er ferdig.

Scrum har hovedsakelig kun tre roller. Dette er "Scrum Master", "Scrum Team" og produkteieren. Produkteieren er personen som innehar en visjon/problemstilling som han ønsker å få oppfylt. Teamet som skal jobbe med selve løsningen kalles Scrum Team. Scrum-teamet har også en oppgave om å holde kontakt med produkteier slik at kunnskapen om både produkteiers forretning og teknologi forenes sammen. Dette vil også sørge for kunden i større grad blir tilfreds med det endelige systemet. For at alt dette rammeverket skal fungere optimalt har en person også rollen som Scrum master. Scrum masteren sin oppgave er å sørge for at teamet følger Scrum slik det er tiltenkt. Scrum masteren vil også være den personen som vil sørge for å fjerne hindringer som kan hindre scrum-teamet i å levere det de skal innenfor den aktuelle sprinten.

3.2.1 Scrum i forbindelse med vårt prosjekt

Scrum Master: Alf Håkon Lille-Mæhlum
Scrum Team: Ingeborg ødegård Oftedal, Erik Frøseth, Tuyet Trang Thi Pham, Tord Sørenes Kloster, Mathias Tempelhaug.

Produkteier: Student-assistent, Espen Skarbsø

I dette prosjektet følger vi dette rammeverket så godt det lar seg gjøre. Etter første møte fant vi ut at alle de ulike timeplanene til gruppemedlemmene gjorde det vanskelig å møtes innen hver 24time for å ha daglige møter. Vi har derfor tilpasset oss så godt det har latt seg gjøre, og har derfor faste møter på mandager/tirsdag kl. 1400-1800 og torsdager kl. 1200-1600. Hver sprint har hatt en varighet på 2 uker, som medfører at vi møtes 4 ganger i løpet av en sprint.

Ingen av oss på gruppa var kjent med Scrum fra før av, så Scrum prosessen vår begynte med at alle begynte å lese hva det gikk ut på. Så begynte vi å lage produktbackloggen. I dette prosjektet har vi ikke hatt en reelprodukteier, noe som medførte at vi ble ekstra selvstendige og nødt til å ta en del valg på egenhånd. Blant annet GUI-design og storypoints i backloggen. Deretter begynte vi å lage vår første sprint ut fra den prioriterte rekkefølgen til produktbackloggen.

Studassen vår Espen har jo fungert som en produkteier. Vi synes det har vært vanskelig å behandle han som en reel kunde, siden han er stud.assen vår samtidig. Vi har hatt møte med han 6 ganger i løpet av prosjektet. På disse møtene har vi spurt han hva han mener er viktig å ta med, og hva han mener er unødvendig. Samtidig har han kommet med tips og råd, blant annet om hvilke verktøy det kan lønne seg å bruke i en slik systemutviklingsprosess. Derfor har møtene med han vært litt udefinerte om det har vært møte med stud.ass eller kunde.

Noe vi syntes var bra med Scrum var at det som faktisk produseres blir synlig. Du vil sikre kvalitet på produktet basert på læring og testing. Vi syntes også at Scrum er en fleksibel modell og det var enkelt og tilpasse seg etter endringer utenfra da det oppstod.

Et av verktøyene vi brukte i forbindelse med å gjennomføre Scrum på en god måte har vært Google Docs. Dette har gitt oss mulighet til å alltid ha oppdatert sprinter og en backlog som vi alle kan redigere når det måtte passe oss. Dette brukte vi dermed for å holde oversikt på hva som ble jobbet med, hva som var blitt gjort og hva som gjenstod av arbeid i den inneværende sprinten. I denne sammenhengen brukte vi også standard diagramverktøy for å fremstille prosessen visuelt i et Burndown - chart. *(Scrum handler mye om tidsestimering, derfor står det mer om dette under punktet tidsestimering 3.3.)*

Andre styrker med Google Docs diskuteres under punktet *(3.4 Verktøy.)*

3.2.2 Productbacklog

Her er vår productbacklog. Dette er en oversikt over alle oppgaver som vi må gjøres for at produktet skal bli ferdig. Oppgavene ligger i prioritert rekkefølge med tanke på hvor viktig det er, og hvor lang tid det tar. Vi lagde denne på et av de første møtene vi hadde sammen. Vi har gjort litt om på den under prosjektet. I henhold til Scrum-modellen, skal disse punktene formuleres som ulike ”stories”, eller krav som ønskes at programmet skal oppfylle. Som vist nedenfor, er ikke alle punktene på denne formen da vi fant det svært unaturlig å formulere noen av disse som en ”story”. Underveis da nye behov dukket opp, ble nye stories lagt til etter behov. Her erfarte vi at det kan være vanskelig å se alle behovene tidlig i en arbeidsprosess, og at man må ta en del høyde for eventuelle ting som dukker opp.

STORY #	BESKRIVELSE	RANGERING (1-5)	SPRINT #			
			1	2	3	4
Sprint 1	***Gjøre klart til programmering av klientprogram***					
1	Skaffe nødvendig teorigrunnlag om SCRUM og SQL	3	10t			
2	GUI-mockup av de ulike vinduene	4	2t			
3	Sette opp server med SVN, MySQL og backup	3	4t			
4	Sette opp kravspesifikasjon i henhold til oppgave/kunde	5	6t			
5	Rapportarbeid	5	10t			
Sprint 2	***Første visuelle utkast av programmet***					
6	Implementere GUI	5		10t		
7	Implementere kartvisning	4	2t	8t		
8	MySQL/Database-kommunikasjon	4		6t		

9	Andre utkast av GUI	2			4t	
10	Rapportarbeid	5		5t	5t	
Sprint 3	***Implementering av funksjonelle-krav***					
11	Systemet skal oppdatere saueposisjoner automatisk	3			5t	
12	Systemet skal sende ut alarm når en sau blir angrepet	3			2t	
13	En bonde skal kunne redigere sauer	3			3t	
14	"Serverprogram" - håndtering av saumeldinger	3			14t	
15	Rapportarbeid	5			5t	
Sprint 4	***Historikk***					
16	Bonde skal kunne se hvor sauene har vært	5				5t
17	En bonde skal se tidligere helse til en sau	4				5t
18	Testing av programmet	4				4t
19	Ferdiggjøring/Finpuss av GUI	4				4t
20	Rapportarbeid	5				5t
Totalt estimert tid/arbeid:			34	29	38	23

3.3 TIDSESTIMERING

(Les mer i oppsummering av sprinter – vedlegg.)

Metoder for tidsestimering er viktig i et prosjektarbeid. Det er fordi vi skal kunne beregne en passende leveringstid, samt hvor vi må befinne oss i utviklingsløpet til enhver tid. Vi vurderte blant annet å lage et Gant diagram, men etter nærmere diskusjon fant vi ut at vi fikk vist mye av den samme informasjonen i en sprint-backlog og i en produktbacklogg. Eneste vi ikke får med er de ulike avhengighetene mellom hver aktivitet. Dessuten følte vi at et Gant diagram ikke nødvendigvis passer godt sammen med Scrum, fordi Scrum er iterativ modell. Da vi estimerte tid, benyttet vi oss av hverandres kunnskap, samt teknikken kalt "planning poker"/"Scrum poker". Dette innebar at vi satte en rangering fra 1 til 10. Det som fikk lavest rangering ble mindre prioritert enn det med høy. Denne rangeringen sa ikke bare noe om prioritering, men også hvor mye tid hvert punkt det ville kreve.

Dermed gikk vi over, først produktbackloggen, deretter sprint for sprint, og diskuterte hvor mye tid de forskjellige oppgavene ville ta. Vi delte opp produktbackloggen i sprinter, og diskuterte og vurderte estimeringen til hvert punkt. Under diskusjonene, viste det seg ofte at vi burde dele opp

oppgavene i enda mindre deler. Vi hadde noen uenigheter om hvor mye tid/høy prioritet vi kom til å bruke på de forskjellige punktene, da måtte vi ta en gjennomgang og begrunne valgene. Vi ble ofte enige etter og har hørt de forskjellige argumentene.

Dette har gitt oss en god oversikt i ettertid over hva vi bør prioritere, og hvordan vi ligger forhold til tidsskjemaet. For en tydelig fremstilling av dette, benytter vi oss av ett "Burndown chart". Det viser hvor langt vi i gjennomsnitt burde ha kommet på et gitt tidspunkt, sammenlignet med hvor langt vi faktisk har kommet.

Tiden vi har satt av hver uke er tid vi vet vi kan jobbe, men vi har tilpasset det etter hvor vi befinner oss i prosessen sett ut i fra burndown diagrammet. Vi har beregnet at hvert enkelt gruppemedlem har jobbet effektivt 8-12 timer i løpet av hver sprint.

3.4 Verktøy

Vi har brukt en del ulike programvare/verktøy i denne utviklingsprosessen. I dette avsnittet skal vi diskutere hva slags verktøy vi har brukt, og fordeler og (eventuelle) ulemper ved disse.

- Subversion (SVN)

I en utviklingsprosess der flere utviklere er involvert, er det svært nødvendig å ha et versjonskontrollsystem. Et versjonskontrollsystem sørger for at alle sammen jobber med siste versjon av kode, og håndterer konflikter hvis to personer endrer på den samme fila samtidig. Alle filene i prosjektet ligger lagret på en sentral server som alle må koble seg opp mot for å hente ut filer, samt lagre endringer som har blitt gjort. Man kan se hvem som har gjort hvilke endringer, og skulle det komme en endring som gjør at systemet slutter å fungere, kan man enkelt gå tilbake en eller flere versjoner.

I og med at gruppa hadde begrenset erfaring med versjonskontrollsystem, fikk vi anbefalt Subversion (SVN) av studentassistent da dette var et av de enklere versjonskontrollsystemene. Vi fikk satt opp serveren tidlig i utviklingsprosessen, og alle i gruppa fikk konfigurert dette sånn at alle kunne bidra med kode tidlig i prosessen. I ettertid har vi sett på dette som et godt valg, da vi fikk det opp å kjøre veldig kjapt. I tillegg har vi hørt at flere andre grupper har hatt litt problemer med andre alternativer som blant annet GitHub.

- NetBeans

Selv om de aller fleste i gruppa hadde mest erfaring med Eclipse som utviklingsplattform, valgte vi å gå for NetBeans. Hovedårsaken til dette valget var at NetBeans tilbyr en GUI-designer som forenkler hele prosessen med å lage et grensesnitt. Dette har spart gruppa for mye tid, samtidig som at sluttresultatet trolig ser bedre ut. I tillegg har NetBeans stort sett de samme funksjonalitetene som Eclipse, så en overgang fra Eclipse til NetBeans tar ikke lang tid. Enda et

argument som talte for NetBeans er at det har en integrert SVN-klient som er svært enkel å bruke. Denne klienten håndterer også eventuelle konflikter og lignende i SVN.

- MySQL

Valget av database var enkelt for gruppa vår av flere årsaker. Først og fremst så var MySQL det databasesystemet som gruppa hadde mest erfaring med. I tillegg ble det anbefalt av faglærer, det er gratis, godt utprøvd og rimelig enkel å bruke. Det ble tidlig i prosessen oppnevnt en database-sjef, som hadde som oppgave å opprette og vedlikeholde databasen. Dette inkluderte også regelmessig backup i tilfelle systemkrasj eller andre uforutsette hendelser. Vi valgte å plassere databasen på en server med tilgang "utenifra". Med dette menes at man kan koble seg opp til denne, uten å være i samme nettverk. Dette forenklet en del av utviklingen, da alle hele tiden jobber mot siste versjon av databasen

- Google Docs

For å holde orden på de mange dokumentene vi har hatt gjennom prosjektet, tok vi i bruk Google Docs da dette gir god støtte for deling og samarbeid. Man er hele tiden sikker på at man har den siste versjonen av dokumentet, og flere kan endre på det samme dokumentet samtidig. I tillegg har Google Docs en god endringslogg, som tillater å angre endringer som har blitt gjort. Eksempler på dokumenter som gruppa opprettet i Google Docs er oversikt over sprintene, tidlige utkast og skisser av systemet, tekstlige Use Case-scenarioer, møtereferater, innloggingsinformasjon til SVN og database osv. En siste fordel er at Google Docs har en enkel versjonskontroll innebygd, som gjør at tidligere versjoner av filer blir bevart. Dette er viktig i et samarbeidsprosjekt, da det fort kan hende at noen overskriver hverandres filer og arbeid.

- Facebook/E-post

Sosiale medier ble tatt i bruk for kommunikasjon innad i gruppa. Det ble tidlig opprettet en privat Facebook-gruppe for å avtale møtetidspunkt og gi andre beskjeder til gruppa utenfor møtene. Facebook ble valgt til dette formålet fordi alle i gruppa er hyppig på Facebook, så meldinger som kommer dit vil raskt bli oppfattet av alle. Epost ble også tatt i bruk noen ganger for å sende ut fellesbeskjeder av ulike slag, men i all hovedsak ble Facebook-gruppa brukt for kommunikasjon.

- OpenMap API (http://openmap.bbn.com/)

For å få inn vising av kart i programmet, har vi valgt å bruke et API som heter OpenMap. Dette er et fleksibelt API, som enkelt lar deg legge ulike lag/layers oppå hverandre. Det nederste laget er bilde som lastes ned i fra Statens Kartverk via et WMS-kall. Mer teknisk informasjon rundt dette, kommer under arkitekturforklaringen. På toppen av dette har vi lagt et lag som viser alle sauene. Hvorvidt dette var et godt valg, er vi usikre på. Da vi har begrenset med erfaring innenfor bruk av kart i Java, er vi usikre på om det finnes andre gode alternativ. Kartløsningen fungerte ikke så godt som vi hadde håpet, så det er mulig vi kunne ha brukt et annet API som hadde gitt et bedre resultat.

- yED Graph Editor

Ulike diagrammer som vi har lagd (eks. UML-diagrammer), er generert ved hjelp av yED Graph Editor. Vi valgte denne da det er en gratis editor og det er enkelt og hurtig å få til diagrammer.

- Wireframe Sketcher

For å lage enkle og bra GUI-skisser, har vi tatt i bruk Wireframe Sketcher. Her kan man enkelt lage fine skisser av GUI. Denne kan lastes ned som en plugin til Eclipse, der man da kan dra inn ulike Swing-elementer for å få skissen til å ligne så mye som mulig på det endelige produktet. Blant annet så lagde vi skisser til å ha i loggen vår, for ikke å glemme hvordan vi hadde tenkt at programmet skulle se ut.

- Office Word

Vi brukte word for å håndtere tekst, tabell og bilder i rapporten. Hele layoutene i rapporten er laget av office word. Grunnen til at vi brukte dette verktøyet, er fordi de forbedrede funksjonene gjør det enkelt å opprette, redigere og få tilgang til dokumenter fra nesten hvor som helst.

4 RISIKOANALYSE

RISIKOANALYSE FOR MEDLEMMENE			
ID #	RISIKOER	SANNSYNLIGHET	KONSEKVENS
1	Fraværende personell	Moderat	Alvorlig
2	Splitting av gruppe	Lav	Alvorlig
3	Kommunikasjon: flere jobber på samme fil	Moderat	Alvorlig
4	Programvarer svikter	Moderat	Tålelig
5	Personlige konflikter	Lav	Tålelig
6	Misforstått prosjektbeskrivelse	Lav	Tålelig
7	Feil estimert tidsforbruk	Moderat	Tålelig
8	Systemsvikt medfører tapt av data	Lav	Alvorlig
9	Lavere kunnskapsnivå enn forventet	Høy	Tålelig

4.1 RISIKOPLANLEGGING

- #1.

  Avtale ny tid og tidsfrister
- #2.

Kontakte hverandre og planlegger ny timeplan

- #3.
  Passe på å oppdatere hverandres fil, før man skal redigere/slette.
  Fordeler eller vite hva de andre jobbe med mot slutten av Scrum-møter
- #4.
  Gjennom opprette system eller lignende.
- #5.
  Ha toleranse for andres meninger.
  Rasjonelle og farlig diskusjoner.
- #6.
  Være sikker på at alle har lest og forstått problemstillingen.

  Jevnlig møter med veileder.
- #7.
  Planlegge iterasjoner, prøve tenke langsikt.
- #8.
  Jevnlig lokal backup, og opplastning til versjonskontroll.
- #9.
  Skaffe hjelp av gruppemedlemmer.
  Spør studentassistent om informasjon.

5 TESTING

Under tidlig utvikling av programmet har vi benyttet oss av forskjellige enkle metoder for å teste javaklassene våre. Et eksempel på en slik test er en *saueTestKlasse*, der vi testet klassen Sau. Denne klassen laget mange instanser av Sau og testet at vi fikk ut svar som forventet. Det vi ville fram til her var å se om man for eksempel kunne lage instanser av sauer uten id, med et ugyldig navn osv. og hvis noe galt skjedde skal man kaste et *Exception*. Vi har også brukt slike testklasser til å teste andre klasser, f.eks*. MySQL_Funksjoner* klassen for å se om *queryet* er skrevet riktig, eller hva som i så fall er galt.

5.1 BRUKBARHETSTESTING

Den første brukbarhetstestene vi gjorde var å sjekke vår prototype. Dette var for å unngå å måtte re-designe brukergrense-snittet senere, samtidig som vi får et inntrykk av hvor godt vårt brukergrensesnitt er. Vi tok et ark og festet noen Post-it lapper på, som forestilte knapper og

lignende i programmet. Deretter testet vi dette på forskjellige personer. Dette gav oss mye nyttig informasjon om hvor det lønte seg å sette forskjellige knapper, hva vi burde ha som overskrifter osv. Det ble allerede i denne fasen avslørt en del svakheter med grensesnittet som vi her klarte å korrigere. Dette inkluderer blant annet plassering av knapper, bruk av farger og lignende.

Videre brukbarhetstesting gjorde vi da vi nesten var ferdig med programmet. Dette igjen, fortalte oss hvordan brukergrensesnittet vårt er, og hva vi burde gjøre annerledes for at det skal bli enda lettere å lære seg programmet. Eksempel på hva vi fikk ut av en av testene, var at vi måtte bytte om på to knapper når en skulle lagre en sau. Vi har lagt ved et vedlegg av SUS, som er et skjema vi har bedt brukeren som har testet programmet fylle ut. *(Se sus-skjema vedlegg).*

Et problem vi har når det gjelder brukerbarhetstesting, er at vi ikke har noen realistiske brukere å teste programmet på. De to personene som testet programmet befant seg heller ikke i de riktige omgivelsene. Optimalt sett skulle vi ha testet systemet på personer med lav IT-erfaring i et miljø som tilsvarer en bondes hverdag (hjemme på kontoret). Observasjonene nedenfor, bygger på resultatene vi fikk fra testene med medstudenter. På en annen side, vet medstudenter i informatikk hvilke problemer og designfeil man bør unngå, så det er uansett svært nytting informasjon vi fikk fra disse testene.

5.1.1 Observasjoner

#	Kategori	Tilbakemelding	Løsning
1	**Legg til sau**	Vanskelig å se hvor knappen "bekreft", når det ligger helt øverst.	Flytte knappen "bekreft" ned, hvor utfylle skjema avsluttes.
2	**Kart**	For å leter etter en bestemt sau er det vanskelig å bla i kartet.	La inn funksjon "Vis sau på kart". Med bestemt posisjonene til sauen.
3	**Hovedvindu**	Etter å ha lukket av kart vinduet, ligger hovedvindu til venstre.	Sette hovedvindu til sentrum etter å ha lukket kart vinduet.

5.2 SYSTEMTESTING

(Se vedlegg for systemtesting punkter)

Systemtesting er viktig for å sjekke om programmet stemmer overens med kravspesifikasjonene og at alle funksjonalitetene er intakte. Testene er designet for å finne feil og for å hindre at noe uventet skal oppstå som kan skape problemer for brukeren. Dette innebærer feil input i felt, uriktig verifisering av input, at alt som skal gå automatisk ikke stopper og generelt at programmet

er i en brukbar tilstand.

Det finnes også gjenværende risikoer som gjør igjen i alle testene:

5.2.1 Maskinvarefeil

A. Risiko: Svært Lav
   1. Finn en annen database
   2. Finn en annen klient-pc

5.2.2 Avbrudd i tilkoblingen

B. Risiko: Lav-middels
   1. Koble til på nytt
   2. Dersom det er noe feil i programmet blir dette håndtert og rapportert.

5.3 DIMENSJONERING

For å sørge for at kravet om dimensjonering er møtt, har vi lagt inn en del testdata i den fungerende databasen *(tilkoblingsinformasjon beskrevet under "server").* To av kravene sier at systemet skal kunne håndtere 200 bønder, og 10 000 sauer. Derfor er databasen fylt med 200 bønder, som hver har 50 sauer. Dette tilsvarer totalt 10 000 sauer. Brukstester viser at systemet ikke har noe problem med å håndtere disse mengdene med sauer, da det er responsivt og stabilt under disse forholdene.

I tidligere tester har vi også hatt en bonde med 10 000 sauer. Testingene med dette viser at etter hvert som disse sauene har en del gps-informasjon og helseinformasjon i databasen, kan det ta en del tid før SheepFinder får lastet inn alle disse dataene. Våre tester viste at det kunne tok rundt 30 sekunder mellom logg inn-vinduet og hovedvinduet, når hver sau hadde rapportert sin lokasjon og helsestatus 50 ganger. I tillegg ble kartvisningen merkbart tregere når man så i historikken, men langt i fra ubrukelig. Når systemet hadde lastet inn alle dataene, var det utover de nevnte punktene ingen merkbar forsinkelse eller treghet i systemet.

Innloggingsinformasjonen til alle disse testbrukerne, er *"bondeX",* hvor X byttes ut med et tall mellom 1 og 200. Passordet er det samme som brukernavnet. Så eksempel på en gyldig login er:

Brukernavn: bonde45

Passord: bonde45

6 ARKITEKTUR FORKLARING

6.1 OVERSIKT ARKITEKTUR

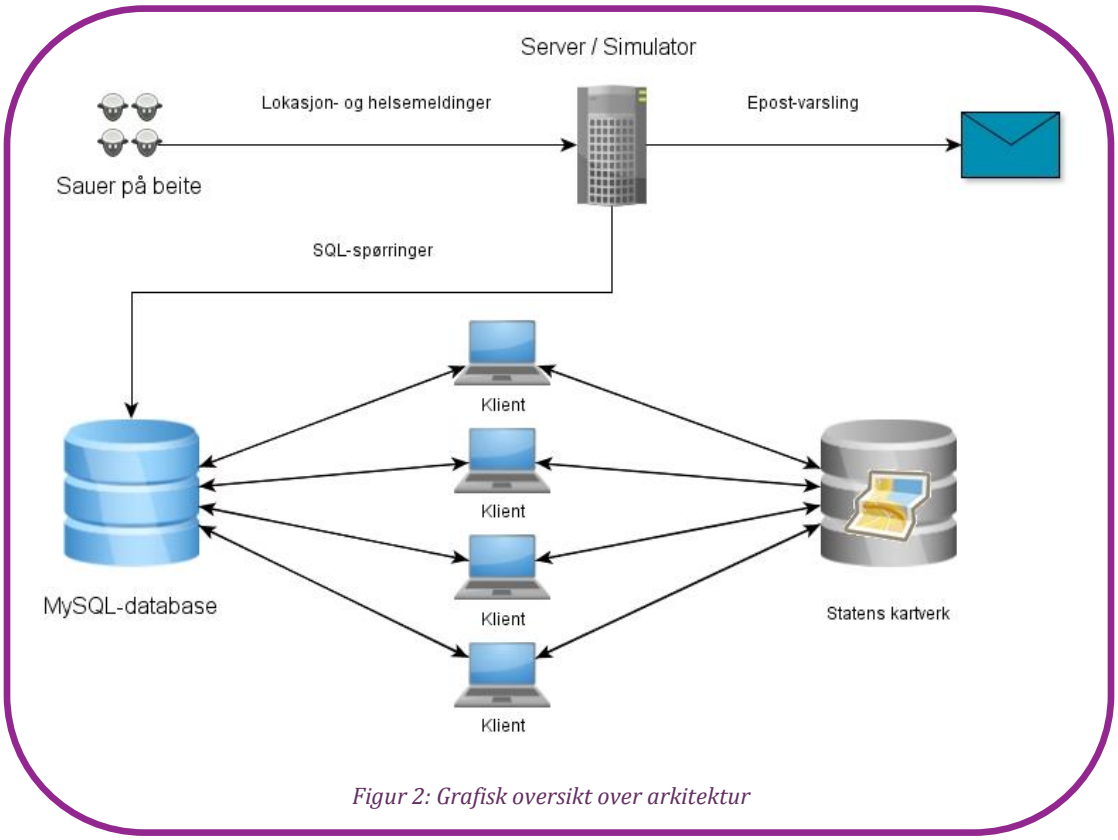
Denne delen har som formål å gi et generelt overblikk over hvordan alle delene i systemet henger sammen, og hvordan disse forholder seg til hverandre.

Sentralt i systemet ligger det en MySQL-database, der all informasjon ligger lagret. Dette inkluderer blant annet informasjon om beitelag og dets bønder, deres sauer med helselogg og posisjonslogg samt informasjon om hvorvidt en sau er under angrep. Klientprogrammet, som vi har valgt å kalle SheepFinder, kobler seg direkte opp mot denne databasen for å presentere all informasjonen på en oversiktlig måte til brukeren. Klienten sender jevnlig ut spørringer til databasen for å sørge for at den har den siste informasjonen presentert til brukeren. I tillegg legger og modifiserer klienten en del av dataene i databasen, som blant annet brukerinformasjon og informasjon om brukerens sauer. Det er et vilkårlig antall klienter som kan koble seg opp mot databasen og kjøre samtidig.

I klienten er det også mulig å se sauenes lokasjoner på et kart. Kartet er et API som heter OpenMap (se avsnittet om verktøy for nærmere beskrivelse). Her hentes kart inn fra Statens Kartverk, via en standard ved navn Web Map Service (WMS). Disse hentes ved at det sendes et URL-kall til en spesifikk adresse med en rekke parametere som sier noe om hvilket kartutsnitt man ønsker å få returnert. Det returnerte utsnittet håndteres så av OpenMap, som sørger for at det blir plassert riktig på kartvisningen og lignende. Under er det vist et eksempel på dette URL-kallet.

```
http://openwms.statkart.no/skwms1/wms.topo2?VERSION=1.1.1&SERVICE=WMS&REQUEST=GetMap&SR
S=EPSG:32633&BBOX=42145.72,6848719.48,98080.18,6893349.59&LAYERS=topo2_WMS&STYLES=&WIDT
H=762&HEIGHT=608&FORMAT=image/png&BGCOLOR=0xFFFFFF&TRANSPARENT=FALSE
```

En svakhet som vi har opplevd ved bruk av OpenMap og WMS, er at kartet virker veldig treigt til tider. Dette er fordi at for hvert kall som kjøres mot Statens Kartverk sine servere, må kartet genereres helt på nytt fra bunnen av. Dette tar noen sekunder, og i tillegg tar det litt tid å sende kartet tilbake til klientprogrammet. Summerer man dette, kan det fort ta opptil fem sekunder før kartet oppdateres. Vi så på Statens Kartverk sin Cache-tjeneste, men fikk ikke til å bruke dette sammen med OpenMap. I samråd med kunde, ble vi enige om at en slik forsinkelse i kartet var OK, da kartet er detaljert og godt.



Figur 2: Grafisk oversikt over arkitektur

En annen viktig del av systemet er serverprogrammet. Serveren har som oppgave å lytte etter meldinger fra sauene som er på beite, for så å prosessere disse. Disse meldingene er lokasjonsinformasjon og helseinformasjon som hver sau sender ut omtrentlig tre ganger i døgnet. Serveren tar inn disse meldingene, analyserer den og setter inn informasjonen i databasen. Hvis dette er en alarm-melding som indikerer at sauen er under angrep, vil serveren også sende ut en epost til bonden som er koblet opp mot denne sauen. Det vil til enhver tid være kun ÉN instans av serverprogrammet som kan kjøre til enhver tid. Hvis det er flere instanser av serveren som kjører samtidig, kan man få problemer med duplikater meldinger i databasen, epostvarsler som sendes ut flere ganger og lignende.

Merk at serverdelen har vi ikke implementert i vår løsning, da dette ikke var en del av oppgavekravet. Vi har allikevel tenkt igjennom hvordan dette bør implementeres i et eventuelt komplett system. Som et substitutt for denne delen samt for å få testet ut klienten, har vi lagd et simulatorprogram. Simulatoren utfører de samme oppgavene som en eventuell server ville ha gjort i systemet. Tre ganger i døgnet oppdaterer den alle saueposisjonene, samt sauenes helseinformasjon. I tillegg vil den ganske sjeldent tilfeldig plukket ut en sau til å være angrepet. Samtidig som den markerer en sau som angrepet, sender den ut en epost til bonden som eier denne sauen.

For at SheepFinder skal være så responsivt som mulig, har vi valgt å ha en kopi av all data som ligger på MySQL-serveren liggende lokalt på hver SheepFinder-klient. Det vil si at når man logger inn som en spesifikk bonde, vil all informasjon i databasen som tilhører denne bonden bli lastet inn lokalt i klientprogrammet. Dette inkluderer informasjon om sauer, helsehistorikk og posisjonslogg. Dette gjør at systemet ikke trenger å sende en forespørsel til databasen hver gang en sau skal vises, eller hver gang en bonde blar i historikken.

For å sørge for at systemet hele tiden har siste informasjon liggende, ligger det en timer og sjekker for oppdateringer i databasen. For å dramatisk minke trafikken, har vi gjort et par grep. De tabellene som blir størst i databasen, er *"health_data"* og *"gps_data".* Disse vil hele tiden vokse, så lenge det er sauer som sender ut informasjon til systemet. Disse tabellene har en primærnøkkel som består av bare en kolonne, som er markert med *auto_increment.* Det vil si at for hver rad som settes inn i disse tabellene, vil primærnøkkelen til den nye raden alltid være en høyere enn den forrige. Når SheepFinder da skal hente ut nye gps-data, husker den hva den siste primærnøkkelverdien den hentet ut var. Det gjør at man kan formulere SQL-spørringen slik at man kan hente ut kun de nye radene som ikke ligger lagret lokalt. Hvis den siste bakgrunnsoppdateringen hentet ut en ny rad med primærnøkkel *4152,* vil den neste SQL-spørringen bli formulert som følgende:

*SELECT * FROM gps_data WHERE gps_data_id > 4152*

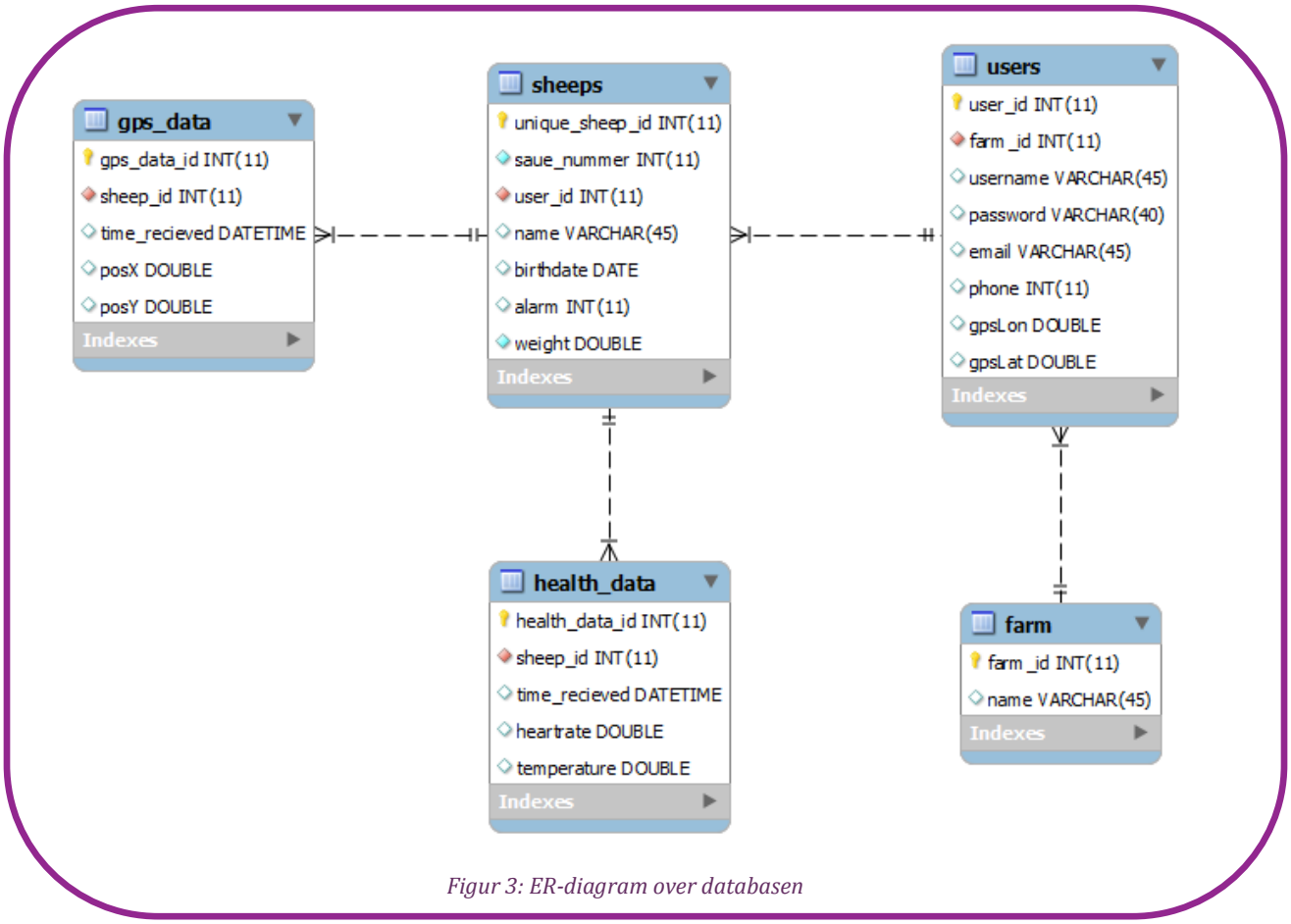
Akkurat den samme teknikken bruker for tabellen *"health_data".* Slik sparer systemet store mengder trafikk.

Denne bakgrunnsoppdateringen henter også ut alle sauene i databasen, og sammenligner disse med den lokale kopien av informasjonen. Hvis det er noen endringer i databasen, vil da disse også finne sted på klienten.

6.2 DATABASE

Dette avsnittet forklarer hvordan databasen er bygd opp, konfigurert og relasjonene mellom de ulike entitetene.

6.2.1 Relasjoner



Figur 3: ER-diagram over databasen

Da systemet er designet for å håndtere flere beitelag, har vi en entitet med navn "*farm*" som representerer et beitelag. Hvert beitelag har en eller flere bønder, som er representert med entiteten "*users*". Denne entiteten inneholder også innloggingsinformasjon for å komme seg inn på" SheepFinder" og endel annen informasjon. For å ivareta brukernes sikkerhet, har vi valgt å lagre passordene kryptert i databasen. MySQL tilbyr flere funksjoner for kryptering av tekststrenger, og vi har valgt å bruke SHA1-metoden da dette anbefales flere steder. SHA1 tar inn en tekststreng og returnerer en kryptert streng på 40 tegn. Det som gjør at denne metoden er

trygg, er at denne krypterte strengen kan ikke reverseres, så det er så og si umulig å dekryptere passordene i databasen.

Til hver bonde, er det koblet en eller flere sauer. Hver sau er registrert med både en unik identifikator, og et sauenummer. Tanken bak dette er at det ikke kan eksistere to sauer med samme identifikator, men to sauer hos forskjellige bønder kan ha samme sauenummer. Entiteten sau har også en egen attributt med navn "*alarm*", som indikerer om en sau er under angrep eller ikke. Dette gjør at man hurtig og effektivt kan sjekke i databasen om det er sauer under angrep. For å imøtekomme kravet med historikk, har vi to tabeller som heter "*gps_data*" og "*health_data*". I disse lagres alle meldinger som blir sendt fra sauene, sånn at man har muligheten til å gå tilbake i tid.

6.2.2 Teknisk/Konfigurasjon

Som nevnt tidligere, har vi valgt å bruke MySQL da den både er gratis, godt utprøvd og et stabilt databasesystem. En annen fordel med MySQL, er at man kan velge ulike lagringsmotorer for de forskjellige tabellene, slik at databasen jobber mest optimalt etter brukerens behov. Vi har valgt å bruke motoren Innom DB på alle tabellene, da dette per dags dato er den eneste motoren som gir full ACID-støtte. ACID er et sett med prinsipper og krav som gjør at dataene i en database er konsistente og er robuste mot feil. Dette gjør også at man kan sette opp fremmednøkler i databasen, som er med på å opprettholde konsistente data. Et eksempel på fordel med dette er at hvis en sau slettes i fra databasen, slettes også sauens tilhørende posisjonsmeldinger og helseinformasjon. I en database uten slike fremmednøkler, hadde disse meldingene blitt liggende igjen til ingen nytte. Et annet eksempel er at en posisjonsmelding kan ikke settes inn i databasen, uten at posisjonens tilhørende sau eksisterer i databasen.

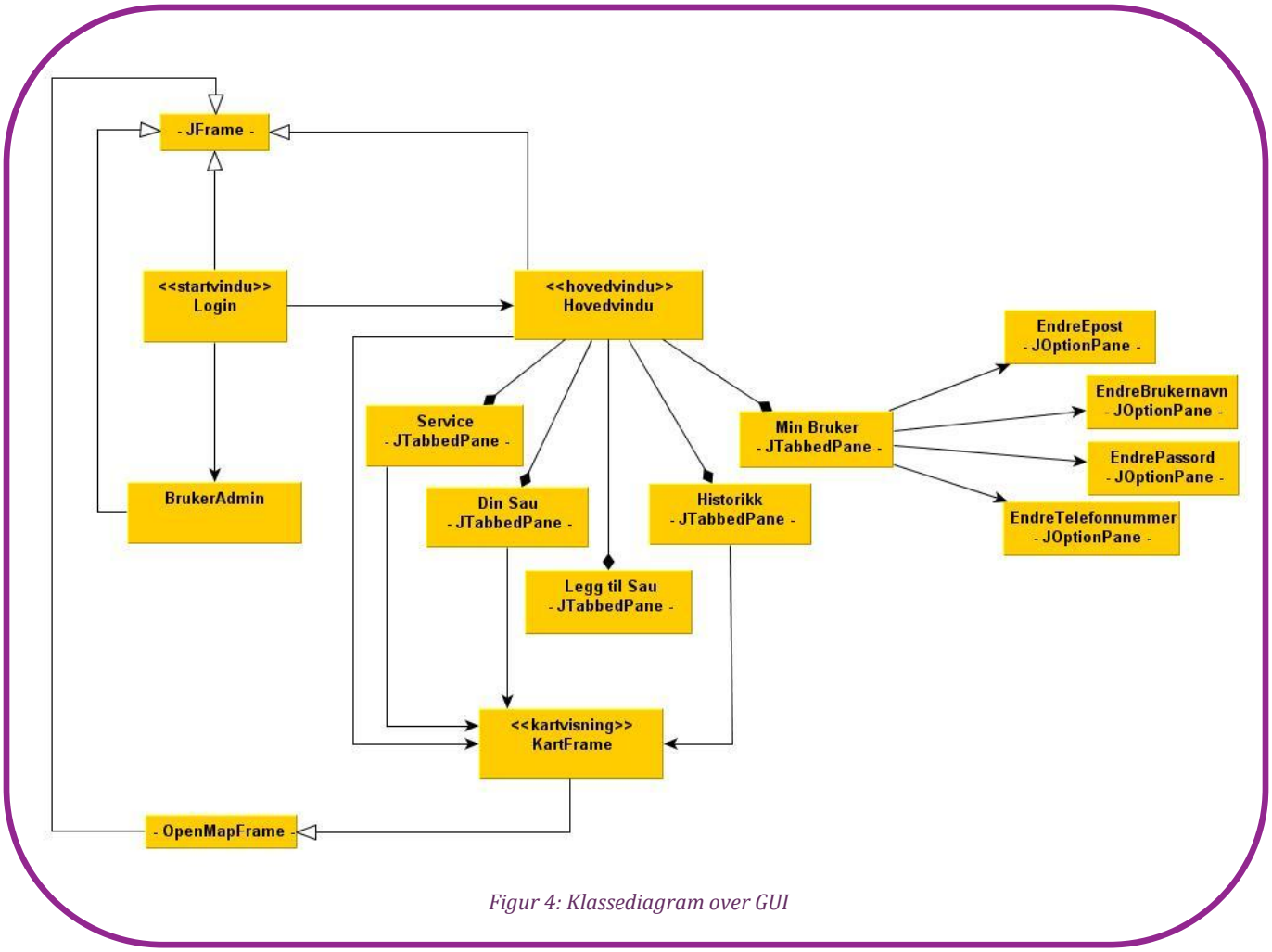
Det er også satt opp et par restriksjoner i databasen, også kjent som" unike indekser". Den første av dem er at to bønder/brukere ikke kan ha samme innloggingsnavn. En annen begrensning er at en bonde kan ikke ha to sauer med samme sauenummer. Men to bønder kan ha sauer med samme nummer. Ved å sette opp slike unike indekser i databasen, er man godt sikret mot tastefeil/brukerfeil og lignende.

6.3 Design

Tidlig i design, og under utviklingen av skisser, fant vi fort ut at det var mye informasjon som skulle kunne vises frem til brukeren. Dette gjorde at vi veldig kjapt bestemte oss for å dele opp brukergrensesnittet i faner, slik at vi fikk delt opp all informasjonen etter hva brukeren faktisk ønsker å se. *(Se mer skisse vedlegg)* Dette gjør det hele mer oversiktlig og enklere for brukeren, enn om vi skulle ha vist all informasjon i ett og samme vindu. Faner har også blitt en mer og mer vanlig layouttype både på mobil og desktop, så en bruker vil sannsynligvis fort kjenne seg igjen ved bruk av faner. Kartet valgte vi å implementere i ett eget vindu. Dette gjorde vi siden dette gjør at brukeren alltid skal kunne se kartet og hvor sauene befinner seg, om han ønsker dette. Hadde vi

valgt å ha kartet i en egen fane, kunne vi ikke vist forskjellig ønskelig informasjon samtidig som vi viser kartet og hvor hver enkelt sau befinner seg. Gjennomgående i programmet har vi bevisst brukt bestemte farger for å tydeliggjøre hva som skjer eller fremheve informasjon. Dette gjelder for eksempel grønne knapper for å lagre endringer, rød tekst på sau med alarmer, rød knapp for slett sau og lignende. Selve utviklingen ble utført i Netbeans, og i implementeringen av GUI elementer brukte vi dermed de ferdiglagde elementene som NetBeans tilbyr.

(Navn på hvert element, plassering og bilder av brukergrensesnittet finnes i vedlegget GUI-diagram.)



Figur 4: Klassediagram over GUI

7 SERVER

7.2 MYSQL - *SHEEPFINDER*

Innloggingsinformasjon til MySQL-serveren.

Adresse	85.89.23.9
Brukernavn	sheepfinder
Passord	gruppe2
Database/skjema	sheepfinder

7.3 SVN VERSJONSKONTROLL - *SHEEPFINDER*

Her ligger det repository for prosjektet.

Brukernavn	svnuser
Passord	gruppe2
URL	svn://85.89.23.9/var/svn/sheepfinder

7.4 SVN VERSJONSKONTROLL – *SHEEPFINDER SIMULATOR*

Her ligger simulatoren vi har brukt som erstatter for reelle sauemeldinger. Merk at dette er et svært lite program, som er lite dokumentert. Likevel skal koden være selvforklarende, og simulatoren er lagt ved her for dokumenteringens skyld

Brukernavn	svnuser
Passord	gruppe2
URL	svn://85.89.23.9/var/svn/sheepfinder_simulator

7.5 BRUK AV SVN I NETBEANS

7.5.1 Checkout/hent prosjekt i Netbeans

1) Gå til menyen *Team* → *Subversion* → *Checkout*

2) Skriv inn URL, brukernavn og passord, og trykk *Next*.

3) Når checkout er ferdig, får du spørsmål om du vil åpne prosjektet. Velg "*Open Project*".

7.5.2 Begynner å bruke Netbeans

- Før du begynner å editere noe i prosjektet, bør du se om det har kommet inn noen endringer. Dette gjør du enkelt ved å høyreklikker på sheepfinder-prosjektet i "*Projects*" - vinduet, og velger *Subversion* → *Update*. Hvis du ikke ser "*Projects*" - vinduet, gå til menyen *Window* og velg Projects for å få det fram.

- Etter at du er ferdig med å gjøre endringer, må du "commite" endringene, altså lagre dem. Gjør dette ved å høyreklikk på prosjektet, og velg *Subversion* → *Commit*. Skriv inn under "Commit message" kort hvilke endringer som er gjort, og sørg for at alle filene er valgt. Trykk så ""*commit*", og vips så er det i boks!

7.6 Bruk av TortoiseSVN

TortoiseSVN er en SVN-klient for Windows.

7.6.1 Installasjon TorroiseSVN

- Last ned og installer TortoiseSVN her: http://tortoisesvn.net/downloads.html

Når det er installert, vil det komme opp mange nye valg når man høyreklikker på mapper, filer og tomme områder i Windows Utforsker. For å få tak i en "repository", gjør man følgende (en repository er det samme som et prosjekt):

1) Høyreklikk på et tomt område i Windows Utforsker der du vil lagre prosjektet. Dette vil typisk være i workspace-mappa til Eclipse.

2) Velg "*SVN Checkout*".

3) Skriv inn riktig URL under "*URL of repository*". Resten skal være valgt riktig på forhånd.

4) Trykk OK og skriv inn riktig "*brukernavn og passord*" når du får spørsmål om det.

5) Deretter er det bare å gå inn og editere/legge til/slette filer. Når en endring har blitt gjort, høyreklikker du på mappa til repositorien, og velger "*SVN Commit*". Skriv gjerne inn en kommentar med hvilke endringer som er gjort.

6) For å se etter endringer, høyreklikk på mappa til repositorien og velg "*SVN Update*".

7.7 Innloggingsinformasjon SheepFinder

Som nevnt tidligere i rapporten, har vi lagt inn en del testdata i databasen for å vise at systemet vårt klarer å håndtere de mengder med informasjon som beskrevet i oppgavekravene. Det er lagt inn 200 bønder, hvor hver bonde har 50 sauer. Brukernavnet til hver bonde er *"bondeX"*, hvor X byttes ut med et tall mellom 1 og 200. Gyldig innloggingsinformasjon vil da være blant annet:

Brukernavn	Passord
bonde12	bonde12
bonde157	bonde157
bonde2	Bonde2

7.8 Oversikt vedlegg

7.8.1 Dokumenter

01 Oppsummering av sprinter

02 Kravspesifikasjon

03 GUI diagram

04 UML diagrammer

05 Systemtesting

06 Brukermanual

07 System Usability Scale - skjema

08 Møtereferat

09 Skisse

10 Videreutvikling

7.8.2 Programvedlegg

- _SheepFinder_javaDoc.zip
- _SheepFinder_kildeKode.zip
- _SheepFinder.jar
- _Sheepfinder Simulator – kildekode.zip
- Sheepfinder Simulator – JAR (katalog med JAR-fil og nødvendige bibliotek)