IT1901 - Informatikk prosjektarbeid I

Gruppe 12
Høst 2013
NTNU

Eivind Morch
Ingrid Watnedal Myrann
Kjersti Fagerholt
Magnus Nermark
Paul Philip Mitchell

20. november 2013

Innhold

1 Introduksjon **1**
- 1.1 Prosjektet . . . 1
- 1.2 Faget . . . 1
- 1.3 Problemstilling . . . 1

2 Prosessbeskrivelse **1**
- 2.1 Gruppen . . . 1
  - 2.1.1 Eivind Morch . . . 1
  - 2.1.2 Ingrid Watnedal Myrann . . . 2
  - 2.1.3 Kjersti Fagerholt . . . 2
  - 2.1.4 Magnus Nermark . . . 2
  - 2.1.5 Paul Philip Mitchell . . . 2
- 2.2 Møter og arbeidsrutiner . . . 2
- 2.3 Scrum . . . 3
- 2.4 Roller i Scrum . . . 5
- 2.5 Tidsestimering . . . 5
- 2.6 Milepæler . . . 7
- 2.7 Evaluering av sprinter . . . 7
- 2.8 Hvordan Scrum fungerte for oss . . . 8

3 Risikoanalyse **8**
- 3.1 Risikohensyn . . . 8

4 Produktbeskrivelse **11**

5 Produktplanlegging **11**
- 5.1 Kravspesifikasjon . . . 11
- 5.2 Arkitekturplanlegging . . . 12
- 5.3 Systemplanlegging . . . 13
- 5.4 Brukergrensesnittplanlegging . . . 14

6 Arkitekturforklaring **14**
- 6.1 Databasen . . . 15
- 6.2 SheepTrackerFX . . . 17
- 6.3 Serveren . . . 17

7 Systemforklaring **18**
- 7.1 MySQL . . . 18
- 7.2 Kryptering . . . 18
- 7.3 Kartet . . . 18
- 7.4 Simulering . . . 19

8 Brukergrensesnittforklaring **19**
- 8.1 Bruks- og brukeranalyse . . . 19
- 8.2 Funksjonalitetsoppsett . . . 20
- 8.3 GUI-design . . . 21

9 Testing **21**
9.1 Brukbarhetstester 22
9.2 Systemtester 22

10 Konklusjon **24**

11 Kilder **25**

1 Introduksjon

1.1 Prosjektet

Dette prosjektet baserer seg på oppgaven som er utgitt i IT1901 - Informatikk prosjektarbeid I, ved Norges Tekniske og Naturvitenskaplige Universitet i Trondheim høsten 2013. Prosjektet har blitt gjennomført som en del av bachelorgraden til alle fem av deltagerne. Hovedmålet for dette prosjektet er for deltagerne å oppnå et innblikk i større gruppeprosjekter og arbeidsmetoder hvor en tredjepart/kunden er en viktig del av prosessen. Ved å programmere i Java får vi samtidig god erfaring i programmeringsspråket og andre verktøy vi har brukt. Vi lærer også å dokumentere koding og arbeidsprosesser, samtidig som vi får anledning til å setter oss dypere inn i databaser og Scrum.

1.2 Faget

IT1901 – Informatikk prosjektarbeid 1 er et fag ved NTNU hvor studentene blir delt opp i grupper for å gjennomføre et programmeringsprosjekt. Via en produkt-orientert oppgave skal vi lære oss å jobbe i grupper og få praktiske erfaring rundt programmeringsvertøy, arbeidsmodeller og det som skal til for å sette sammen et software-produkt. Hovedmålet er å gi eleven en praktisk tilnærming til teori og metoder rundt systemutvikling.

1.3 Problemstilling

Vi skal utvikle et brukervennlig program som i hovedsak skal fungere som et register og alarmsystem for sauebønder. Bonden ønsker et program som kan gjøre det lettere å administrere sauene sine; holde oversikten over helsetilstander, ha en geografisk oversikt og bli varslet ved et eventuelt angrep. Vi skal lage dette programmet i henhold med kunden/bondens ønsker, hvor hver sau blir registrert i en database og er tilgjengelig for den respektive bonden. Vi som gruppe skal implementere Scrum i vår arbeidsrutine og følge denne metoden ut etterhvert som prosjektet utvikler seg.

2 Prosessbeskrivelse

2.1 Gruppen

2.1.1 Eivind Morch

Ansvarsområder: Scrum-master. GUI. Diagrammer. Generell koding.

Kunnskap: God kompetanse på kode og programstrukturering. Har en del erfaring med grafisk arbeid. Er også god på organisering og strukturering.

2.1.2 Ingrid Watnedal Myrann

Ansvarsområder: Møtereferat. Rapport. Oppsett i Latex. Diagrammer. Generell koding.

Kunnskap: Har kunnskap rundt programmering i Java og Python. Har noe erfaring med Latex, er god til å skrive og har lyst til å jobbe med rapport i tillegg til programmering.

2.1.3 Kjersti Fagerholt

Ansvarsområder: Rapport. Diagrammer. Produkt backlog-oppdatering. Generell koding.

Kunnskap: Har en del programmeringskompetanse i både Java og Python. Har grunnleggende databasekunnskaper fra videregående og har lyst til å jobbe med dette.

2.1.4 Magnus Nermark

Ansvarsområder: Databasen. Kartsystem. Simulering. Generell koding.

Kunnskap: Har programmeringskunnskaper i python og java. Grunnleggende databasekunnskaper fra videregående. Har holdt på en del med telnet/ssh klienter som PuTTY på fritiden. Ønsker å programmer og jobbe med databasen. Er et strukturert gruppemedlem.

2.1.5 Paul Philip Mitchell

Ansvarsområder: MySQL-metoder. Visning av saue-informasjon. Logg. Generell koding.

Kunnskap: Kan programmere i Java, Python og noe i C#. Har kunnskap rundt databaser, dokumentasjon og diverse utviklingsmodeller. Er også et strukturert gruppemedlem og god til å skrive.

2.2 Møter og arbeidsrutiner

Første kommunikasjon var via mail for å få kontakt med alle deltagerne og for å få fastsatt et møte. Det kom da frem at to av de oppskrevne gruppemedlemmene ikke tok IT1901 dette semesteret, men vi fikk heldigvis med oss Kjersti Fagerholt og var klare for å sette i gang.

Første møte gikk med på å bli bedre kjent, samtidig som vi diskuterte hvilken oppgaver vi mente vi var sterke på og hva vi gjerne ville jobbe med. Scrum-master ble valgt og denne jobben bestemte vi skulle gå til den mest engasjerte i starten, som var Eivind Morch. I oppstartfasen begynte vi grovt å estimere de nødvendige arbeidsoppgavene og sprintøktene, samtidig som vi planla de faste møtetidene. (Se hvordan vi implenterte Scrum i prosjektet vårt under avsnitt 2.3 om Scrum). Som de fleste vet var UKA i løpet av dette

semesteret, og flere av gruppemedlemmene var frivillige som jobbet for UKA både før og under, derfor ble møtetidene litt flyktige. Men om intet annet ble oppgitt ble møtetidene slik:

Mandag 12:00-14:00

Torsdag 11:00-12:00

Mandagsmøtet resulterte ofte i en lengre arbeidsøkt og gikk ut over de fastsatte tidene for de som ønsket. Vi lagde også en egen Facebook-gruppe hvor kommunikasjonen gikk kontinuerlig både rundt møtetider og statusoppdateringer. Etterhvert benyttet vi oss også av Google Calendar som oversikt over møtetidspunkter, se vedlegg 09-Programvare og verktøy. Det kan nevnes at mot slutten av semesteret ble møtetidene hyppigere og vi hadde flere felles arbeidsøkter hvor vi var nøye på å ha full kontroll og at alle hadde nøyaktig oversikt over de andres arbeidsstatus.

Under hvert møte ble det skrevet referat. Første møte begynte vi med å skrive fritt det som ble diskutert og gjennomgått, men etter hvert laget vi en mal i forhold til hvordan det skal gjøres i Scrum. I den malen gikk vi først gikk gjennom hva hver deltager hadde gjort og hvordan vi lå ann i forhold til det som var planlagt. Se vedlegg 02-Møtereferater for et eksempel fra et møte. Så gikk vi gjennom aktuelle punkter og problemer som vi ønsket å diskutere i fellesskap. Disse punktene ble ofte utfylt i møtemalen på forhånd av den som ville ta det opp. Sist gikk vi gjennom arbeidsoppgaver og hva hver deltager skulle gjøre til neste møte. Dersom noen hadde lest seg opp på en teknologi som kunne være relevant for prosjektet, ble dette presentert i løpet av møte og bestemt om det skulle integreres eller ikke.

For å følge med på hvilke oppgaver andre medlemmer i teamet jobbet med laget vi en mal for øktlogg, som vi skulle fylle ut for hver oppgave vi gjorde. Her ble oppgaven beskrevet, startdato fylt inn og problemer som dukket opp underveis ble dokumentert. Se et eksempel på dette i vedlegg 03-Øktloggeksempel. Vi lagde også en oversikt over diverse arbeidsoppgaver og lignende på Checkvist, se vedlegg 09-Programvare og verktøy, hvor alle gruppemedlemmene kunne krysse av etter hvert som vi ble ferdige med deloppgavene. For en illustrasjon av hvordan dette så ut, se vedlegg 07-Checklist.

2.3 Scrum

Scrum er et utviklingsmiljø der tverrfaglige team utvikler produkter eller prosjekter på en iterativ, inkrementell måte. I dette utviklingsmiljøet struktureres utvikling i sykler kalt Sprinter. Disse syklusene har som regel en lengde på omtrent to uker, men aldri mer enn fire uker. Sprintene er såkalt «timeboxed» - de ender på en spesifikk dato selv om arbeidet som skulle ha blitt gjort innen denne dato er ferdig eller ikke, og tidsrommet utvides aldri. Vanligvis velger Scrum Teams én sprintlengde og bruker denne lengden for alle sprintene deres inntil de er i stand til å forbedre seg og kan bruke en kortere sykel. På begynnelsen av hver sprint velger et tverrfaglig team, bestående av rundt syv personer, diverse

elementer (kundens krav) fra en prioritert liste, kalt product backlog. Teamet blir enig om et kollektivt mål de mener de kan levere til kunden ved utgangen av sprinten, noe som er konkret og som vil være fullstendig ferdig – med andre ord ikke en helt fullverdig løsning, snarere kun et fullstendig ferdig modul av hele løsningen. Under sprinten skal ingen nye elementer legges til; Scrum tar hensyn til forandringer til neste sprint, men den foreløpige korte sprinten er ment å fokusere på et lite, klart, relativt stabilt mål. Teamet samles hver dag for et kort «stand-up møte» for å inspisere sin fremgang, og justere de neste stegene som trengs for å fullføre det gjenstående arbeidet. På slutten av sprinten gjennomgår teamet sprinten sin med interessenter, og demonstrerer hva de har bygget. Her får også teamet tilbakemeldinger som kan inkorporeres i neste sprint. Scrum understreker et fungerende produkt på slutten av sprinten som er virkelig «ferdig»; eksempelvis i et tilfelle av et programvareprosjekt, betyr dette et system som er integrert, fullstendig test, dokumentert for sluttbruker og potensielt leverbart.

Å implementere Scrum i prosjektarbeid i korte trekk:

- Velge Scrum-Master.
- Lage produkt backlog, prioriter oppgavene.
- Avtale varighet på sprinter i gruppa.
- Sette mål for hver sprint.
- Gruppa ser på hvert element i produkt backlog og vurder hva disse oppgavene innebærer. Del opp i mindre oppgaver. Gjør dem klar for å deles inn i sprinter.
- Dele opp alle oppgavene i sprinter.
- Planlegg sprintene i detalj: Estimer tid for hver oppgave(Bestem start- og sluttdato). Bestem hvem som har ansvaret for hver oppgave.
- Starte sprinter, følge oppsatt sprint.
- Oppdatere produkt backlog om nødvendig.
- Følge utviklingen på et Burndown chart. Mål: At teamet når null timer/dager av sum arbeidstimer når sprinten er ferdig.
- Hyppige møter med kunde. Hovedsakelig etter hver sprint.
- Sprint evaluering: Vise utkast av produktet til hele teamet og produkteier/kunden. Vurder: Har teamet levert det som skulle vært levert? Dette hjelper medlemmer av teamet å være fokusert på fristene- siden ingen vil møte opp på et sprint evalueringsmøte uten noe som helst. Vise utkast av produktet til hele teamet og produkteier/kunden. Tilbakemelding fra kunde
- Tiltak for å gjøre arbeidsprosessen og produktet bedre.

2.4 Roller i Scrum

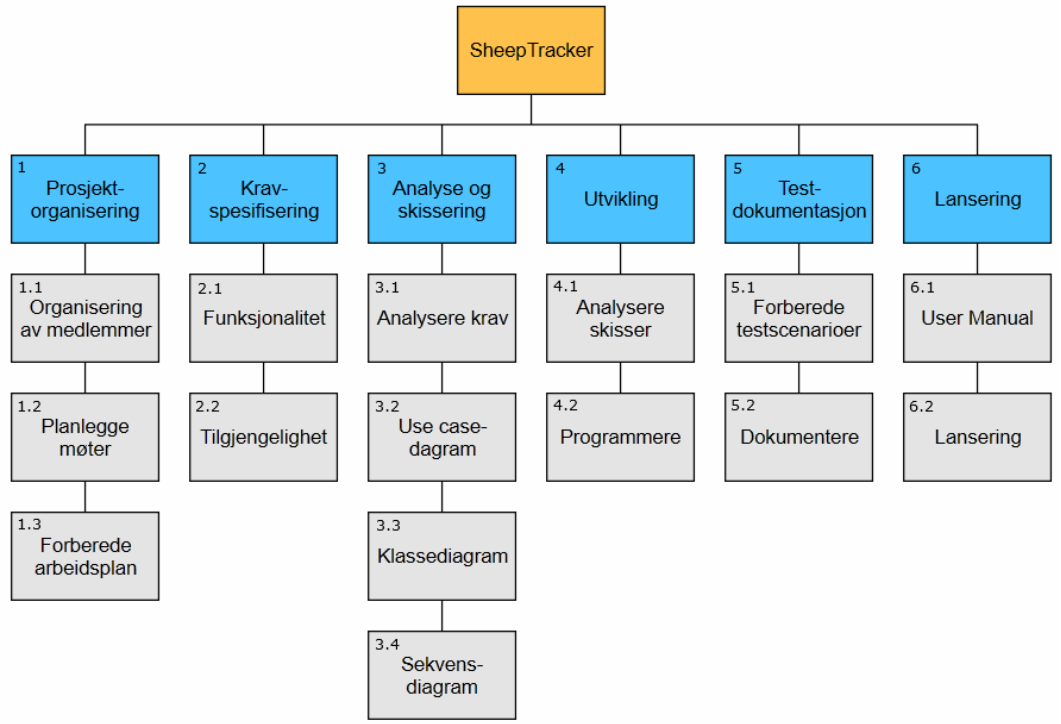
Produkteier: I vårt tilfelle er det ingen spesifikk produkteier som vi har regelmessige møter med – studassen vil være det nærmeste vi kommer til å inneha denne rollen, derfor vil denne rapporten heretter omtale henne som kunden". Teamet vil ha hyppige møter med kunden slik at hun kan vurdere kvaliteten av arbeidet som er blitt gjort. I en vanlig Scrummodell vil også produkteieren har ansvaret for å presentere produktfunksjoner, sette disse inn i en prioritert liste og kontinuerlig opprettholde og oppdatere den prioriterte listen. I vårt prosjekt er produkt backloggen vi går laget fra oppgaveteksten som følger med prosjektet. Denne blir endret på for å kunne dytte inn flere funksjoner, eventuelt fjerne overflødige funksjoner. Teamet vil selv komme med eventuelle nye ideer eller krav til produktet.

Teamet: I vårt tilfelle er vi fem personer med relativ lik bakgrunn og erfaring, så vi delegerte arbeidsoppgaver så fornuftig vi kunne og prøvde å rullere arbeidsoppgaver mellom hver sprint, slik at alle medlemmer av teamet får erfaring i alle aspekter av prosjektet. Teamet planlegger sammen hvordan produkt backlog og sprinter skal bygges opp, og hvordan de best kan gjennomføre prosjektet. Teamet jobbet tett sammen for å kune oppnå målet de har satt.

ScrumMaster: I små scrumteam, som i vår prosjektoppgave, er det vanlig å la et av teammedlemmene fungere som en Scrum Master, men som da har ansvar for en mindre del av teamets kollektive arbeidsoppgaver. Det var også slik vår Scrum-Master fungerte. Vi slo fast at vår Scrum Master sine oppgaver var å guide teamet i korrekt bruk av scrummodellen, som for eksempel inneholdt å sørge for at møtene ble holdt riktig, produkt backlog og risikolisten ble oppdatert. Samtidig har Scrum Masteren vår fungert som kommunikasjonsledd mellom prosjektgruppen og kunden.

2.5 Tidsestimering

På oppstartsmøtene satt teamet opp en produkt backlog. Se vedlegg 11-Produkt Backlog. En produkt-backlog skal i hovedsak inneholde alle krav som produkteier har til produktet. Produkteieren har i følge Scrummodellen ansvar for produkt-backlogen, og bestemmer prioriteten til oppgavene som teamet skal utføre. I vårt tilfelle ble det til at gruppa sammen gikk gjennom oppgaven vi hadde fått utdelt av faglærer og skrev ned de kravene. Se avsnitt 5.1 om kravspesifikasjon for nærmere beskrivelse. Vi måtte oppfylle for å kunne fullføre prosjektet, som gjaldt både produktet og rapporten. Deretter diskuterte vi prioriteten til de forskjellige oppgavene og rangerte dem med tall fra 1-5, i tillegg til at vi ga hver oppgave poeng(1-10) etter hvor vanskelige vi trodde de var. (I scrum-modellen blir dette kalt story points). Vi laget også en oversikt kalt Work Breakdown Structure(WBS), som ligger i vedlegg 14-Work Breakdown Structure. WBS ga oss en bedre oversikt over prosjektets sammenheng og fremgang. I figur 1 vises en komprimert versjon av WBS.



Figur 1: Komprimert work breakdown structure.

Neste steg var å bestemme varighet på sprint og sette opp sprint backlog, se vedlegg 01-Sprinter, som er en mer detaljert oversikt over hva som skal gjøres og når de skal gjøres. Gruppen ble enige om at hver sprint skulle vare i to uker. I vedlegget for sprinter er alle sprintene delt opp og presentert sammen med burndown chart. Oppgavene fra produkt backlog ble delt inn i sprinter - en tidsperiode på to uker, der hver sprint representerer en større del av prosjektet som skulle bli ferdig, og som et nytt utkast vi kunne vise til produkteier. Oppgavene fikk underpunkter med flere detaljer og det ble satt hvor lang tid oppgavene skulle ta, estimert av teamet.

Hver oppgave hadde en oppstartsdato og estimeringen av tiden ble gjort i dager. Det ble enklere for oss å estimere i dager framfor timer, siden vi likevel måtte dele opp å gjøre større oppgaver på flere dager, og fordi vi i tillegg ofte sa at oppgaver skulle gjøres til neste møte.

Underveis måtte vi reestimere tiden på noen av oppgavene, men når disse oppgavene var utført skrev vi i sprint backlog den reelle tiden vi hadde brukt på oppgaven. Med et burndown chart for hver sprint kunne vi følge med på hvordan utviklingen av prosjektet var i forhold til det estimerte tidsskjemaet. Se Vedlegg 01-Sprinter for en framstilling av dette. I burndown chart er tidslinjen på den horisontale aksen og summen av den estimerte tiden på den vertikale aksen, slik at det etter hver dag vises hvor mye arbeid som er igjen av sprinten.

Vi lagde også et Gantt-diagram som baserte seg på produkt backlogen vår. Diagrammet illustrerer prosjektets tidsskjema slik at vi kan se hvilke oppgaver som skal gjøres i hvilken rekkefølge og hvordan disse ligger i forhold til hverandre. Her vises også hvilke «milestones», eller tidsfrister, som teamet har jobbet mot. Se vedlegg 13-Gantt-diagram.

I motsetning til sprint-backlogen, så brukte vi ikke Gantt-diagrammet like effektivt. Derfor ble ikke denne oppdatert underveis på samme måte, da det tok mye tid å endre på den i verktøyet Ganttproject, se vedlegg 09-Programmer og verktøy, og vi mente det uansett ikke var så nyttig for teamet siden vi i stedet brukte backlogen.

2.6 Milepæler

Teamet hadde flere frister å forholde seg til underveis i prosjektet. Disse milepælene er også markert i Gantt-diagrammet vårt (se vedlegg 13-Gantt-diagram).

17.09.13 Avslutning første sprint - All planlegging skal være klart.

26.09.13 Møte med kunden. Presenterer første utkast og får tilbakemelding.

01.10.13 Avslutning av andre sprint. Registrering og innlogging av bonde skal være ferdig.

15.10.13 Avslutning av tredje sprint. Registrering av sauer skal være ferdig, samt planlegging av midtveispresentasjon.

16.11.13 Midtveispresentasjonen.

29.10.13 Avslutning av fjerde sprint. Kart, simuleringer, alarmer og loggføring skal være ferdig.

12.11.13 Avslutning av femte sprint. All koding skal være ferdig.

14.11.13 Det ferdige produktet vises til kunden.

19.11.13 Avslutning sprint seks. Rapport og testing er ferdig. Sluttpresentasjonen holdes

20.11.13 Innlevering av rapport.

2.7 Evaluering av sprinter

I enkelte tilfeller brukte vi mindre tid enn planlagt på sprintene og i enda flere tilfeller brukte vi mer tid som naturligvis skapte forsinkelser. Noen oppgaver tok lengre tid enn forventet blant annet fordi vi ikke regnet med nok tid til å hente inn informasjon om diverse områder, som databaser og kartimplementering. Vi oppdaget også endringer som måtte gjøres i implementeringer som ble gjort i forrige sprint, og at oppgaver ikke ble gjort ferdig til riktig tid, pga. lite effektiv jobbing. I tillegg hadde gruppa også, som forventet og antatt i risikoanalysen, noen fravær grunnet sykdom. Se mer om dette under avsnittet 3.1 om risikohensyn.

På slutten av hver sprint ble det gjort en evaluering, og i forbindelse med dette presenterte vi vårt utkast av prosjektet til kunden og fikk tilbakemeldinger av henne. Etter at teamet hadde tatt en evaluering av sprinten, og hentet ut

eventuelle erfaringer, ble det satt i gang tiltak for å forbedre produktet og arbeidsprosessen. Tips fra kunden gjorde ofte koden vår mindre kompleks og et eksempel på tiltak for å bedre arbeidsprosessen var å endre tidspunkt for møter for å minke fraværet i gruppa. Gruppa ble også enig om å skjerpe effektiviteten, fordi oppgavene måtte bli ferdig til rett tid. På denne måten ble produktet og arbeidsprosessen til gruppa forbedret for hver sprint.

2.8 Hvordan Scrum fungerte for oss

Vi hadde en positiv opplevelse med å bruke Scrum. Vi opplevde hele hensikten med Scrum å være at vi hele tiden kunne ha full oversikt over prosjektet og prosjektets framgang. Problemer som dukket opp underveis ble raskt oppdaget og tatt hånd om. I Scrum prosessen har utviklerteamet ofte kontakt med kunden, i vårt tilfelle stud.ass, og presenterer det man har for ham/henne. Dermed blir det mulig å gjøre endringer i krav, i motsetning til for eksempel fossefallsmetoden der krav er fastsatt på forhånd. En slik presentasjon for produkteier øker produktiviteten, fordi det motiverer gruppa til å ha et nytt utkast klart for presentasjon etter hver sprint.

3 Risikoanalyse

Ved prosjektets begynnelse fant vi frem til mulige risikofaktorer som kunne være til hinder eller gjøre prosjektet vanskeligere for oss. I begynnelsen kom vi frem til flere ulike ting som kunne gå feil. Felles for disse var at flere av dem baserte seg på elektroniske feil eller feil i programkoden. Når man lager en risikoanalyse over et prosjekt som dette er det viktig å oppdatere listen av eventuelle risikoer etter hvert som vi blir kjent med arbeidsmengden og arbeidsmetoden. Noe som ble tydelig utover i semesteret var at den største forsinkelsesfaktoren var menneskelige feil. Det var sjelden at forsinkelser oppstod av feil i det elektroniske. Dermed oppdaterte vi risikolisten med flere former for menneskelige feil, inkludert de vi hadde erfart og mulige feil vi antok mulige fremover i prosjektet. Dette er vist i en forenklet modell av risikolisten i tabell 1. For en mer detaljert oversikt med de faktiske kalkuleringene, se vedlegg 10-Risikoliste.

3.1 Risikohensyn

#1 Beskrivelse: Elektronisk datatrøbbel kan komme i form av tapt kode, PC som krasjer, problemer med commit, update i NetBeans, o.s.v.

Preventive tiltak: All data må lagres som backup. Før en commiter den nyskrevne koden er det viktig at en har oppdatert til siste versjon av programmet.

Skadehåndtering: Når noe skjer er det viktig å varsle de andre gruppemedlemmene så tidlig som mulig. Vi hadde kun mindre problemer med kodingen, men heldigvis var det bare å gå tilbake i commit-loggen for å finne frem til det vi mistet.

Risikoliste			
ID #	**Beskrivelse**	**Sannsynlighet**	**Konsekvens**
1	Datatrøbbel, elektronisk	Lav	Høy
2	Mangel på tid	Lav	Høy
3	Misforståelser	Middels	Middels
4	Sykdom	Høy	Lav
5	Mangel på kompetanse	Høy	Lav
6	Datatrøbbel, menneskelige	Høy	Lav
7	Dårlig oppmøte	Lav	Høy
8	Krangling/uenigheter	Lav	Middels

Tabell 1: Forenklet risikoanalyse

#2 Beskrivelse: Manglende tid kan forekomme ved flere stadier. Både ved hver sprint og mot slutten av prosjektet. Det kan også hende at vi har feilestimert størrelsen på aktiviteten.

Preventive tiltak: Siden vi alle er nye ved Scrum og aldri har jobbet i grupper ved slike store prosjekter må vi regne med mye feilestimering av tid. Derfor må vi gi oss selv god tid rundt de forskjellige delene av prosjektet.

Skadehåndtering: Vi oppdaget dette gradvis utover prosjektet. Av og til måtte vi forkaste tidskrevende ideer, andre ganger måtte vi ta oss selv i nakkeskinnet og jobbe hardere. Og atter andre ganger lå feilen i tidsestimeringer, hvor vi bare var nødt til å bruke mer tid på den aktuelle sprinten.

#3 Beskrivelse: Misforståelser inkluderer alt fra at gruppen har misstolket oppgaven til at det er kommunikasjonssvikt mellom gruppemedlemmene.

Preventive tiltak: Det er viktig at vi setter oss ned i gruppe og analyserer oppgaven, samt legger frem vår forståelse av den. Vi må også ha en åpen diskusjon med kunden, som i dette tilfellet er vår stud.ass.

Skadehåndtering: Om dette skulle skje må vi få til et møte hvor vi kommer frem til den riktige tolkningen. Om aktiviteten allerede er startet med grunnlag i missforståelser må vi se ann hvor mye som kan brukes og hva som må forkastes. Samtidig må vi ta tidsrammene med i beregningen.

#4 Beskrivelse: Ved sykdom mener vi i så stor grad at en deltager blir gjentakene fraværende på møter og ikke får gjennomført sine arbeidsoppgaver.

Preventive tiltak: Det er viktig at alle medlemmene både lever sunt og ikke sliter seg ut. Det er også viktig at det blir varslet tidlig om noen begynner å bli syk, slik at de andre medlemmene kan forberede seg på at noe av dens oppgaver kanskje ikke blir gjennomført på tiden.

Skadehåndtering: Når en av gruppemedlemmene har blitt syke kan det være nødvendig å delegere de mest essensielle arbeidsoppgavene videre til andre som har tid.

#5 Beskrivelse: Manglende kompetanse er noe som lett kan oppstå når man jobber med slike store prosjekter for første gang.

Preventive tiltak: Som gruppe må vi sette av mye tid for å lese oss opp og lære oss om de delene vi ikke kan og tenger å vite for ferdigproduktets skyld.

Skadehåndtering: I vårt tilfelle måtte vi for eksempel henlegge den første planen om å bruke Statens Kartverk ved GPS-lokasjonen til sauene. Det ble til at vi måtte finne en enklere løsning som kanskje er noe mindre ideell or kunden. Altså ved manglende kompetanse er det viktig å innse tidlig når noe er for vanskelig eller tidkrevende på et tidlig stadium, slik at vi ikke kaster bort for mye tid på det.

#6 Beskrivelse: Ved datatrøbbel i form av menneskelige feil er det snakk om hvordan vi ofte som uerfarne kodere kan gjøre noe feil, både i selve koden og i de programmene som vi bruker rundt kodingen.

Preventive tiltak: Et godt samarbeid, hvor alle blir opplært i rutiner, samt dobbeltsjekking, er det viktig å både jobbe nøye for å unngå og sette av tid i tilfelle det skulle forekomme. Alle deltagerne må også sjekke sin egen kode ofte.

Skadehåndtering: Om det skulle skje at noen har gjort noe feil ved vårt system må den personen varsle de andre tidlig, slik at ikke alle bygger sin koding videre på dette. Samtidig må feilen spores opp og rettes på så best det er mulig.

#7 Beskrivelse: Det kan hende at vi får dårlig oppmøte på møtene, både grunnet sykdom, missforståelser og feil ved møtetid/-sted. Samtidig kan det hende at noen er for travle til å dukke opp.

Preventive tiltak: For å forhindre dette er det viktig at alle forstår alvoret i møtene, og er åpen for å ha en god dialog med de andre. Møtetidene må utarbeides i fellesskap slik at alle kan og vet når de er.

Skadehåndtering: Om det er noen som gjentatte ganger ikke møter opp må Scrum-master eller noen andre i gruppen ta tak i personen og fortelle den hvordan de andre ser på situasjonen.

#8 Beskrivelse: Uenigheter og dårlig stemning blant gruppemedlemmene er en mulighet med tanken på både faglige uenigheter, gruppens samarbeid og personlige situasjoner.

Preventive tiltak: Gruppen må ta avgjørelser i plenum og Scrum-master har ansvar for at alt går diplomatisk for seg. Alle må oppføre seg profesjonelt og tenke på prosjektets beste.

Skadehåndtering: Om uenigheter vedvarer må Scrum-master initiere en gruppesamtale.

4 Produktbeskrivelse

Teamet vårt skal lage et program for bønder som skal kunne registrere sine sauer i en database. Formålet med dette er at bonden enkelt skal kunne oppdatere seg på informasjon om sauene sine som vekt, helsestatus, lokasjon og eventuelt om sauen er under angrep. Programmet kan kjøres på bondens datamaskin via filen SheepTracker.jar og det krever at Java er installert på datamaskinen.

Bruker må opprette seg en profil i programmet med e-post og passord, samt annen kontaktinformasjon. I registreringen får bruker opp et kart der det er mulig å søke opp gårdens adresse og få lengde- og breddegradene automatisk registrert til profilen. Det er også mulig å registrere kontaktinfo til en «hjelper» i databasen. «Hjelperen» blir også kontaktet hvis en sau er under angrep eller i fare. For en mer detaljert beskrivelse av fremgangsmåten, se vedlegg 08-Brukermanual.

Når bonden logger seg inn kan han registrere nye sauer, og disse får en startposisjon som er den samme som gården til bonden. Bonden kan få presentert et kart med lokasjonen til alle sauene i flokken som oppdateres og loggføres i hver enkelt sau sin posisjonslogg. I en historielogg får bonden presentert de viktigste oppdateringer som har skjedd siden forrige innlogging, som kan være endringer av helsetilstand eller eventuelle angrep. Sauer som har blitt lagt til nylig, blir også registrert her.

Om en sau skulle være under angrep, eller i annen fare, vil SheepTrackerFX motta et signal om dette og sende ut varsler via SMS, oppringing og e-post. Hvis bonden har registrert en «hjelper» i databasen vil også denne bli varslet på samme måte. For en mer teknisk forklaring av de forskjellige funksjonene, se avsnitt 7 Systemforklaring.

5 Produktplanlegging

Dette avsnittet vil gå gjennom planleggingsfasen av selve produktet. Dette gjelder både arkitekturen, systemet og brukergrensesnittet. Alle disse punktene representerer viktige deler av vårt produkt som hadde behov for felles planlegging og designutvikling. Hva vi endte opp med vil bli nærmere forklart i sin helhet under hvert sitt avsnitt lengre ut i rapporten. De antagelser vi har tatt rundt oppgavens krav er beskrevet under hver planleggingsprosess. Først vil vi presentere hva vi la som grunnsteiner for oss selv før vi begav oss ut på planleggingen.

5.1 Kravspesifikasjon

Vi startet tidlig med produktplanlegging. Ved vårt andre gruppemøte gikk vi nøye gjennom oppgaveteksten og medfølgende krav til systemet i fellesskap. Vi følte at de ikke-funksjonelle kravene som blir gitt i oppgaveteksten er mer relevant for senere i utviklingsfasen, og vi bestemte oss for å utsette gjennomgang av disse til vi hadde et skjelett å bygge videre på, samt en bedre

forståelse av hvordan systemet kom til å fungere. Dermed satt vi heller ikke opp noen prioritering på disse.

Med inspirasjon fra eksempelrapporten og eksempelfunksjonene gitt i oppgaveteksten utarbeidet vi så en del egendefinerte krav til programmet. Dette skulle gi oss en oversikt over arbeidsoppgaver, samt rammer for utvikling av programkode. Kravene vi selv satte er listet i prioritert rekkefølge under.

Brukeren skal kunne -

1. Logge inn
2. Registrere sau
3. Sjekke sauenes sist registrerte lokasjon
4. Sjekke sauenes helsetilstand
5. Gå tilbake i tid og sjekke logger (lokasjon og helse)
6. Registrere bruker
7. Se sauenes lokasjoner på kart
8. Redigere sauinformasjon
9. Slette sau
10. Redigere sin profil

Etter at vi hadde blitt enige om krav til systemet videreførte vi dem til et USE CASE-diagram. Se vedlegg 04-Use case. Use case brukes til å identifisere systemkrav ved å gå gjennom de stegene som kreves for at brukerens mulige interaksjoner med systemet skal bli gjennomført med ønsket utfall. Dette kan gjøres ved felles eller separate visuelle og tekstlige oversikter over interaksjonene. Vi så det tilstrekkelig med et felles diagram for alle use casene, men laget tekstlige representasjoner av alle individuelle krav. For den tekstlige delen brukte vi malen som er brukt i eksempelrapporten ettersom vi følte den ga mulighet til tilstrekkelig, men ikke for detaljert, innhold.

I tillegg til kravene til systemets logikk ønsket vi også å gjøre krav til systemets brukergrensesnitt. Det er viktig å passe på at programmet er brukervennlig for målgruppen. For denne oppgaven antok vi at målgruppens datakompetanse kunne være alt fra liten til svært god, og vi måtte dermed tilpasse brukergrensesnittet slik at det ikke ville kreve mer enn grunnleggende erfaring med datamaskiner, men samtidig inneholde tilstrekkelig funksjonalitet. Les mer om dette under avsnitt 8 brukergrensesnittforklaring.

5.2 Arkitekturplanlegging

Noe av det første vi gjorde i planleggingsfasen var å prøve å se for oss hvordan ulike komponenter i det ferdige systemet som skulle sammenkobles. Dette var

alt i fra hvordan programmet skulle se ut, skulle vi kode i java, hvordan skulle databasestrukturen være og hvilken server skal vi bruke. Vi begynte med å tegne et utkast til hvordan vi så for oss databasestrukturen for programmet. Se figur 2 for hvordan dette ble seende ut til slutt. En skisse over hvordan kommunikasjonen mellom database, klient, sauer og server ble også laget. Vi tolket oppgaven slik at det ikke kreves en faktisk eksisterende server. For å spare tid bestemte vi oss dermed tidlig for å erstatte den med lokale scripts. Dette gjorde at vi måtte gjøre en del antagelser for den teoretiske serverens funksjon, samt enheter som var direkte tilkoblet denne.

Antagelser for sensor på sauer:

1. Sensor på sauene sender informasjon tre ganger per døgn.
2. Sensor på sauene kan oppfatte helsetilstand.
3. Sensor på sauene kan oppfatte lokasjon.
4. Sensor på sauene kan oppfatte angrep.

Antagelser for server:

1. Server kan behandle og analysere informasjon fra sauer.
2. Server kan sende behandlet informasjon til SheepTracker.
3. Server er oppgående og tilgjengelig slik at det tilfredsstiller krav om tilgjengelighet i oppgaveteksten.
4. Server kan motta og behandle informasjon fra over 10000 sauer på en gang.

5.3 Systemplanlegging

I starten av utviklerfasen var det relativt stor usikkerhet i gruppen om hvordan programmet skulle struktureres. Et viktig aspekt i dette prosjektet er jo at vi som utviklere ikke nødvendigvis hadde tilstrekkelig med kompetanse fra begynnelsen av. Dermed var vi forberedt på å måtte gjøre endringer etter hvert som vi lærte. For å få en bedre oversikt begynte vi å skissere klassediagram tidlig. Klassediagram er et diagram som viser interfacer, klasser, inkludert attributter og operasjoner, og deres samhandlinger og relasjoner. Dette brukes ofte tidlig i utviklerprosesser for å få en oversikt og mal over programkodens innhold.

Første idémyldringer ble gjort ved hjelp av skisser på papir, men resultatet av dette ble så lagt inn i et diagram laget på datamaskin. Dette diagrammet, se vedlegg 12.1-Første klassediagram, ble da brukt som en tidlig mal i utviklerfasen. Det tok ikke lang tid før vi så en del feil i det første klassediagrammet, og det ble åpenbart at vi trengte å utarbeide en nyere og bedre versjon for å opprettholde struktur i utviklingen. For å få en bedre forståelse av hvilke prosesser

som kreves av systemet for at kravene vi tidligere hadde satt skulle tilfredsstilles, tok vi i bruk sekvensdiagram, se vedlegg 06-Sekvensdiagram. Sekvensdiagram brukes til å definere og vise interaksjonene mellom objektene i systemet for en gitt situasjon, og sortere disse interaksjonene og prosessene etter tid. Dette brukes også gjerne tidlig i utviklerfasen. Her fant vi at det kun var nødvendig å illustrere et fåtall hendelser ettersom prosessene disse innebærer og valgene vi gjorde under utformingen av dem påvirket flere av de andre hendelsene.

Ved hjelp av sekvensdiagrammet laget vi så et nytt klassediagram. Denne gangen med forbedret erfaring og forståelse, mye større nøyaktighet og flere detaljer. I tillegg hadde vi nå fått bedre kompetanse på database og kommunikasjon mellom database og klient, som gjorde at vi kunne lage et velfungerende skjelett. Det nye diagrammet, se vedlegg 12.2-Andre klassediagram, ble brukt videre i utviklerfasen.

Vi tolket oppgaven slik at det ikke var et krav at SMS faktisk skulle sendes og at det ikke skulle gjøres en faktisk oppringning av brukerens registrerte telefonnummer. For å spare tid bestemte vi oss dermed for at vi skulle erstatte dette med enkle simuleringer og systemmeldinger for demonstrasjon av den teoretiske funksjonen. Vi antok også at brukeren har et norsk mobilnummer for å forenkle implementasjonen.

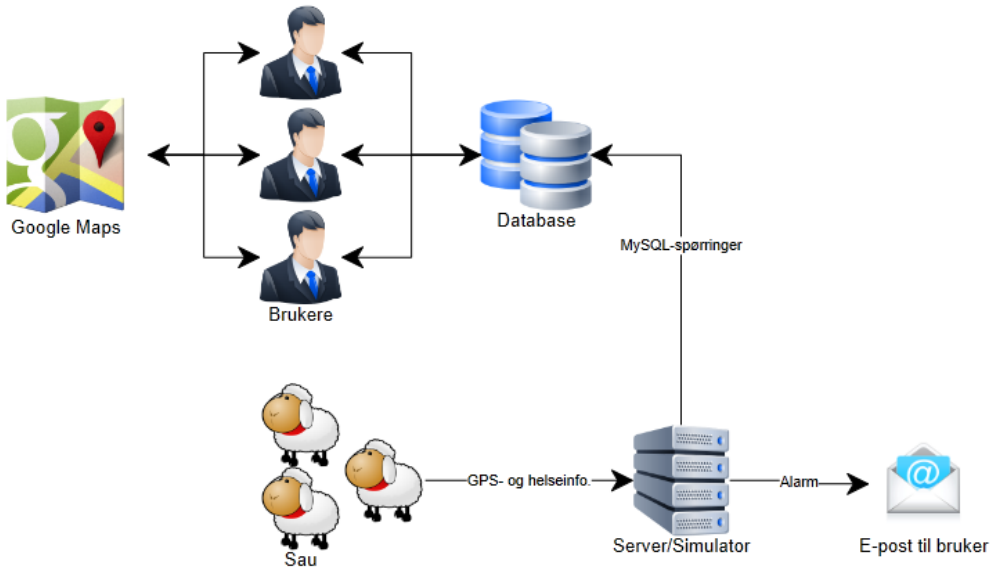
Krav om at bruker skulle kunne se sau på kart tolket vi først som at det kun kreves at det kunne vises én sau på kartet av gangen. Da dette kom opp under møte med kunde ble det uttrykt ønske om en kartfunksjon som kunne vise alle bondens sauer samtidig. Dette resulterte da i en forandring i planlagt kodestruktur, men ikke av noen stor grad. For registrering av gård antok vi at bruker ville kunne peke ut gårdslokasjonen på kartet.

5.4 Brukergrensesnittplanlegging

Vi laget en skisse over hvordan klienten skulle se ut, altså hvordan brukeren selv skulle oppleve programmet. Denne skissen er vist i figur 5. Denne ble laget for å få mer oversikt over funksjonaliteten i programmet. Skissen viste seg senere å være altfor detaljert, og den ble senere forkastet mot en enklere utforming. Forsøket på å lage eget design til programmet ble også gjort, men dette ble også forkastet. Mer om dette i avsnittet 8 om brukergrensesnitt.

6 Arkitekturforklaring

Programmet vårt består av tre hoveddeler. Disse er databasen, serveren og selve programmet slik som figur 2 viser. Dette avsnittet vil nå gå gjennom de forskjellige delene av vårt system.



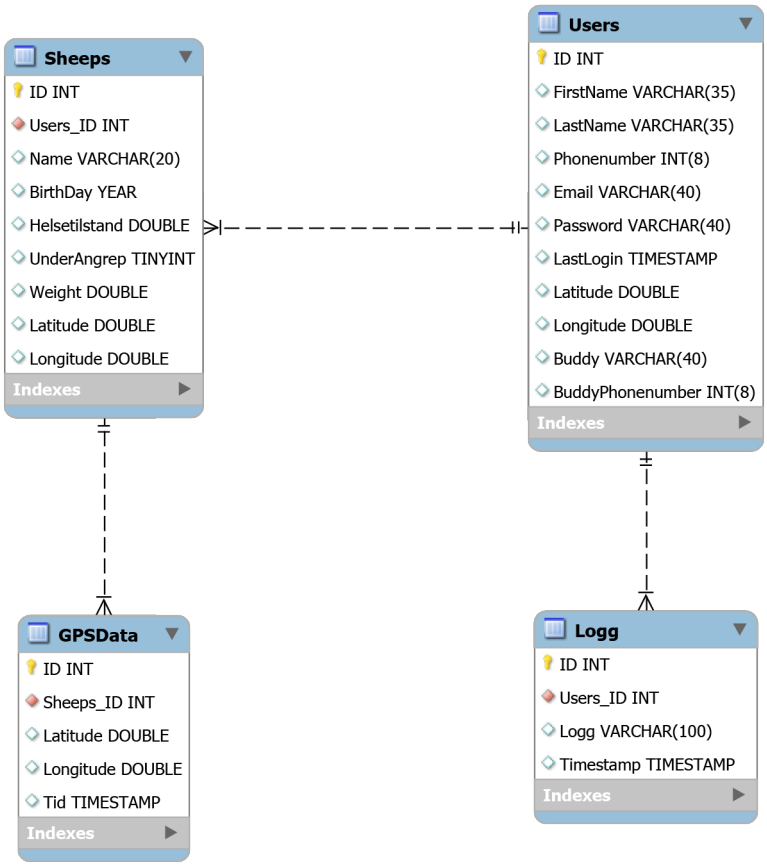
Figur 2: Oversikt over arkitekturen

6.1 Databasen

Programmet vårt er bygget rundt en MySQL-database som er satt opp på NTNU sitt nettverk. Her blir alt av informasjon lagret og det er her programmet vårt finner den informasjonen som brukeren etterspør og endrer. Dette inkluderer blant annet informasjon om hver bruker og hver sau, i hver sin entitet, se figur 3.

Databasen er hver bonde representert som en entitet med navn ”Users”. Her lagres all nødvendig informasjon om hver bonde som brukernavn, e-postadresse, gårdens lokasjon og e-postadressen til den som eventuelt har blitt registrert for å få en kopi av varsel-mailen om en av sauene er under angrep. Passordet for hver bruker er også lagret her, som krypteres ved hjelp av en SHA1-hash. Slik blir passordet lagret som en lengre kryptert streng og er i teorien en trygg og enkel enveis-hash, les mer om dette under avsnitt 7.2.

I tillegg har vi en entitet ved navn ”Sheep” som er koblet opp mot hver bonde ved hjelp av bondens automatiske ID-nummer. Alle entitetene av ”Sheep” som registreres under en viss bonde får automatisk bondens ID-nummer som en egen attributt. Ellers inneholder entiteten generell informasjon om den enkelte sauen. Fødselsdatoen er kun lagret som årstall, ettersom alle sauer er født rundt den samme uken på våren. Dag og måned er derfor overflødig informasjon for bonden. ”Sheep”-entiteten inneholder også en attributt ”UnderAngrep” som forteller hvorvidt sauen er angrepet og om systemet skal sende ut alarm til bonden med sauens lokasjon. Lokasjonen til sauen kommer i form av bredde- og lengdegrader, slik et GPS-system ofte fungerer, og denne blir lagret i en egen entitet som vi har kalt GPSData. I henhold til kundens ønske og oppgavens spesifikasjoner kan bonder dermed se



Figur 3: ER-diagram

den tidligere lokasjonen i form av en logg til hver enkelt sau. Den siste entiteter heter ”Logg” og er en oversikt over den siste handlingen til hver innloggede bruker.

6.2 SheepTrackerFX

Programmet vårt har vi kalt SheepTrackerFX og er brukerens måte å få tilgang til all den informasjonen som databasen inneholder. SheepTrackerFX sender en MySQL-spørring til databasen, databasen gir tilbake det koden spør etter og SheepTrackerFX viser dette til brukeren via GUIen. I tillegg kan brukeren spørre etter en sau sin lokasjon, SheepTrackerFX vil da vise sauens lokasjon på et kart i et nytt panel. Det var denne funksjonen som fikk oss til å gå over fra Java SE til Java FX, les mer om dette i vedlegg 09-Programvare og verktøy. Ved å kjøre Java FX-panelet inne i en applet kan vi bygge en nettleser inne i dette panelet med hjelp av webengine og webview, eller -vindu. I det vinduet blir det kjørt en html-fil med javaSqript som henter inn et Google Maps-API v.3. Med andre ord vil si at programmet åpnet opp et vindu med de kartfunksjonene som Google Maps allerede inneholder og som brukeren kan trenge. Sauen blir vist frem i GUI-en som en liten sau plassert i de bredde- og lengdegradene som sauen sit sendte inn til serveren.

6.3 Serveren

Beskrivelsene under er bygget på antagelsene som ble gjort i planleggingsfasen. Spesifikke forklaringer av disse antagelsene finnes i avsnitt 5.2 Planlegging av arkitektur.

Serveren på sin side er den komponenten i vårt program som har kontakt med både brukeren og sauene. Serveren lytter til det sauene sender inn av data og prosesserer det i henhold til de instruksjonene programmet vårt har satt. Den oppdaterer lokasjonen til den enkelte sauen, får inn den nyeste helseinformasjonen og hvorvidt sauen er under angrep. Om sauen viser seg å være under angrep vil serveren sende ut en mail til brukeren og en til den mailadressen som brukeren har satt opp under "hjelpersystemet". Den vil også sende en sms til det nummeret brukeren har gitt. Vårt program er derimot ikke koblet opp mot en faktisk server ettersom dette ikke var en del av kravene for denne oppgaven. Serverens oppgaver blir demonstrert av en egen simuleringsklasse. Simuleringen vår er hardkodet som en egen Java-klasse som går rett inn i databasen for å hente og forandre informasjon, og må aktiveres. Simuleringsklassen har egne metoder som oppdaterer sauenes posisjoner og helseinformasjon. Vi har også en metode som itererer gjennom sauene og ved bruk av en random-variabel, plukker ut en av sauene som blir registrert som under angrep. Ved hjelp av en underklasse kaldt myTimer blir disse metodene kjørt med en fastsatt tidsintervall, i vårt tilfelle har vi satt den på 30 sekunder.

7 Systemforklaring

Dette avsnittet vil nå gå gjennom noen av de mest sentrale delene av vårt system. Dette er forskjellige deler som vi synes var utfordrende, som vi brukte mye tid på eller som vi mener fortjener en nærmere forklaring i denne rapporten.

7.1 MySQL

Til lagring av data har vi valgt å bruke MySQL-database. Her ligger blant annet informasjonen om sauene og deres posisjonslogg, samt hvilke sauer som hører til de respektive bøndene. Klienten kommuniserer direkte med databasen ved hjelp av spørringer. Dersom brukeren foretar endringer, for eksempel endring av personlig data, eller oppretting av sau, kjøres spørringer opp mot databasen. MySQL er et gratis og godt utprøvd opensource databaseverktøy. Ettersom NTNU hadde en MySQL server tilgjengelig for studenter, ble valget enkelt.

7.2 Kryptering

Som tidligere nevnt har vi valg MySQL som database. For å ivareta brukerens sikkerhet har vi valgt å bruke en enveishash som kalles SHA1 (secure hash algorithm). Denne tar inn en streng og returnerer en 20-bits hash-streng som blir lagret i databasen. Ettersom det er en enveis hash-funksjon, er det så å si umulig og dekryptere passordet dersom passordet i utgangspunktet er gjennomsnittlig sterkt.

Programmet kjører diverse spørringer opp mot databasen. Første spørring skjer ved innlogging. Her sjekkes brukernavn og passord opp mot databasen ved

```
"SELECT ID FROM Users WHERE Email=? AND Password = SHA1(?)"
```

Tilsvarende spørring skjer ved for eksempel oppretting av sau, eller registrering av ny bruker, hvor en MySQL-streng blir forberedt og sendt til databasen.

For å få ned responstiden til programmet, valgte vi å redusere datatrafikken mellom klienten og databasen. Dette gjorde vi ved å hente all informasjon fra databasen idet programmet initialiseres etter innlogging. En kopi av data fra databasen ligger da lagret i minne til programmet, og hentes fortløpende etter behov. Dette gjør at total ventetid går ned, og flere klienter kan koble seg opp til databasen samtidig uten at det går nevneverdig utover responstiden fra serveren.

7.3 Kartet

Kartet er implementert ved at en nettleser blir bygget inni et JavaFX-applet. På denne måten kunne nettleseren kjøre et egenkomponert HTML-side som laster inn kartet.

HTML-siden kjører primært JavaScript basert på Google Maps API(Application Programming Interface). Til å begynne med så vi på hvordan vi kunne utnytte statens kartverk til vårt kartsystem. Da denne metoden mislyktes, valgte vi å gå over til JavaFX-webengine løsning. Vi valgte å bruke Google sitt API til kartet. Dette fordi det er et stabilt og presist kart med god responstid. API-et er også relativt enkelt å modifisere etter ønsket behov. Se vedlegg 09-Programvare og verktøy.

For å få Java-appletet til å kommunisere med javascript måtte vi lage en bro(Bridge) mellom JavaScriptetet i HTML-filen og nettleseren(WebEngine) i Java-Applet, som sender variabler og metoder mellom kartet og klientprogrammet. Dette var en litt knotete måte å løse det på, men ettersom vi valgte at brukeren selv skulle plassere gården sin ved å klikke/dra markøren, så vi dette som eneste utvei.

7.4 Simulering

Siden serverdelen av programmet ble droppet, valgte vi å gjøre simuleringen lokalt på klienten. Dette gjorde vi med en egen simuleringsklasse som kjører en metode som oppdaterer posisjonen til alle sauer, samt regner ut en randomverdi som vi har valgt å kalle en sjanse for at en sau skal bli angrepet eller at helsetilstanden skal blir endret. Dersom en sau blir angrepet, vil en alarm vises på skjermen, loggen oppdateres, og en epost vil bli sendt. For å sende en epost bruker vi JavaMail-API, som sender en Epost via via SMTP-kobling opp til Google sin epost-server. I Javaklassen SendEmail, blir en SMTP(Simple Mail Transfer Protocol) benyttet for utformingen av eposten, før den sendes til brukeren via Google sin server. Eposten inneholder navnet til sauen som ble angrepet, og en link som viser siste registrerte posisjon til sauen.

8 Brukergrensesnittforklaring

Når man lager programmer for kunder som kun ønsker enkle og spesifikke funksjoner, samtidig som deres kunnskaper rundt teknologien kanskje er minimal, er det viktig å gjøre det ferdige programmet så brukervennlig som mulig. Dette handler om å gjøre de forskjellige funksjonene i programmet så intuitivt og åpenbare som vi klarer. Som it-studenter og som en yngre generasjon er det lett å bli blindt på dette feltet, vi har et annet syn på og tilnærming til teknologi enn hva andre kanskje har.

8.1 Bruks- og brukeranalyse

Fordi vi alle var relativt ukjente med å sette opp GUI-designet, valgte vi å jobbe med den generelle kodingen og få en bedre klasse- og funksjonoversikt før vi startet den faktiske utformingen av brukergrensesnittet. NetBeans tilbyr ferdiglagde GUI-elementer og dette hjalp oss med å forstå og implemendtere GUI-desiget etter hvert som vi kodet. Når vi hadde kommet et stykke med dette satt vi oss ned for å diskutere det sammen. I begynnelsen så

vi for oss at dette var et program som brukeren lett kunne ønske å ha som mobilapplikasjon. Designet på brukergrensersnittet falt da i nærheten av hvordan slike programmer ofte er, nemlig at man har en og en side oppe av gangen og hopper fra den ønskede siden via de knappene man har tilgjengelig, samt frem- og tilbake-knapper. Dette kan sees på figur 5 av førsteutkastet vårt for brukergrensesnittet. Grunnen til dette er at når man har samme program på flere plattformer kan det være ønskelig å gjøre brukergrensesnittet tilnærmet lik. Men ikke langt ut i denne designdiskusjonen gikk det opp for oss at brukeren med all sannsynlighet ikke var av samme aldersgruppe som oss, ei heller samme teknologiinteresse. Derfor henla vi ideen om applikasjonsinspirasjonen og bygget brukergrensersnittet opp på enkel og oversiktlig måte som kunne minne om et fysisk og analogt mappesystem, nemlig faner. Ved å ha hver side som en egen fane vil brukeren få en god oversikt over de forskjellige sidene og de funksjonene de tilbyr uten å måtte huske programoversikten selv. Dette viste seg også å være fordelsmessig med tanken på kjøretiden til programmet.

Vi kjørte også en mindre test på noen vi anså som noenlunde lik målgruppen for programmet og fant flere brukergrensesnitt-elementer som måtte relokaliseres og forenkles, se avsnitt 9.1 for en mer om denne testen. Mer om dette kan leses under brukbarhetstester. Ved å kjøre denne testen håpet vi på å gjøre programmet vårt mer intuitivt rundt de funksjonene som vi selv hadde jobbet lenge med og derfor blitt litt blinde for.

8.2 Funksjonalitetsoppsett

Når man først åpnet programmet har vi valgt å ha login og registrering i to forskjellige faner for å ikke distrahere brukeren med flere felter som ikke er relevant for det han/hun ønsker å gjøre. Når brukeren er logget inn vil han/hun bli møtt av flere fanevalg. Vi har skilt programmets funksjoner under ”min profil”, ”legg til en sau”, ”søk etter sauer” og ”logg”. På den måten vil brukeren alltid ha oversikt over de mest grunnlegende tingene han/hun kan foreta seg.

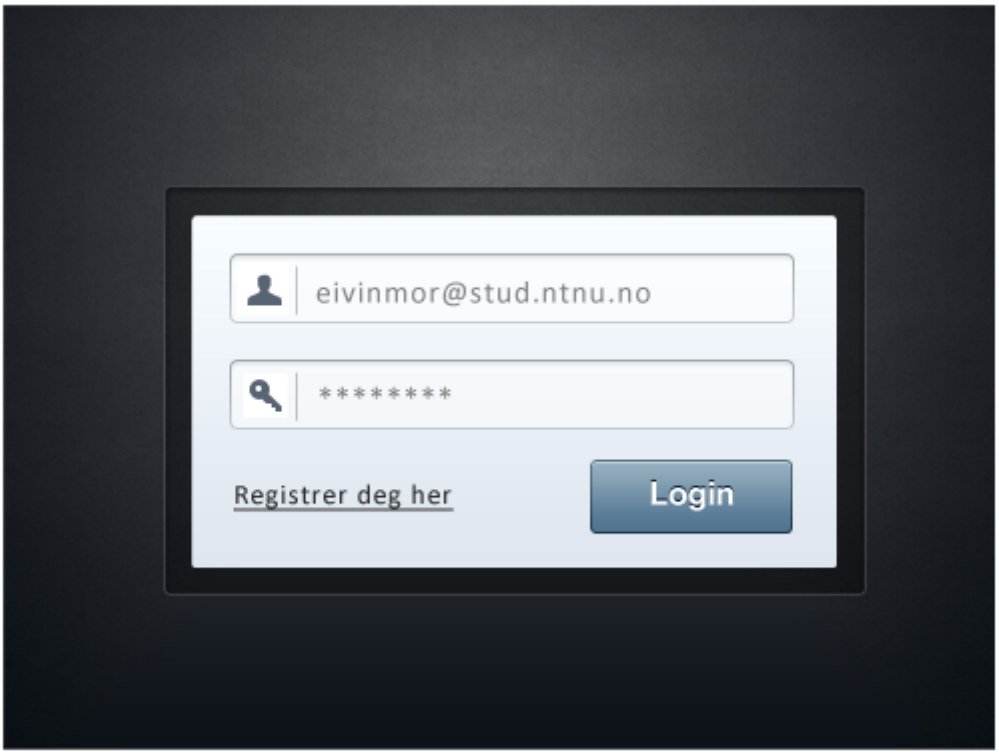
Under ”min profil” kan brukeren se og forandre sin egen informasjon. Alle feltene har en tekst som forklarer hva som skal fylles inn i hver respektive tekstboks, samtidig som det er en illustrasjonstekst i noen av tekstboksene som fungerer som eksempel for hva som skal fylles inn. Det samme gjelder under ”legg til sau”. Under ”søk etter sauer” har vi valgt å vise søkeresultatet i samme vindu som feltet hvor man søker. Når man søker vil mulige resultater dukke opp like under, og når man trykker på dem vil informasjonen dukke opp like til høyre. Mer om dette kan leses og er illustrert i vedlegg 08-Brukermanual. Det som er den mest essensielle informasjonen, nemlig helsetilstanden, er fargekodet. Er den grønn er sauen utenfor fareområdet. Er den gul vil det være småproblemer. Blir den oransje kan det være nødvendig med tilsyn og er den rød er sauens tilstand kritisk. Dette er en fargeskala som vi kulturelt sett er vandt til at illustrerer alvor eller klarsignaler.

Om brukeren ønsker å sette gården på kartet eller se lokasjonen til en eller flere sauer, vil dette dukke opp i et nytt bilde. Slik kan brukeren alltid ha kartet oppe etter hvert som lokasjoner oppdateres og brukeren kan enkelt følge med

på sauene sine samtidig som han/hun gjør andre aktiviteter på datamaskinen og i programmet.

8.3 GUI-design

Vi hadde intensjoner om å oppdatere GUI-designet når vi var helt ferdig med kodingen. Vi lagde et utkast til dette i PhotoShop som er illustrert i figur 4. Dette tok lang tid og å implementere det i koden tok enda lengre tid. Etter en samtale med kunden ble vi fortalt at et slikt design ikke var noe som var spesielt ønsket eller ville bli lagt fokus på. Kunden ønsket det så enkelt som mulig. Derfor forkastet vi dette og sparte mye tid på å bare legge til noen enkle bilder av en søt sau og noe gress i bakgrunnen, samtidig som vi i hovedsak brukte NetBeans sitt design. Kunden bekreftet at dette var tilstrekkelig.



Figur 4: Forkastet GUI-design

9 Testing

Under utviklingen av SheepTrackerFX har vi benyttet oss av flere forskjellige former for tester. Da vi var ferdige med et hvert modul av programmet, bestemte vi oss for ikke å bruke Junit tester, da ingen hadde særlig erfaring med det og vi fant ut at det er enklere og like effektivt å bare teste i det kjørende programmet. Den typiske fremgangsmåten vi benyttet oss av illustreres best med vinduet for å registrere en bruker – her har vi flere former for validering,

blant annet at mail skal være på korrekt format og at passordene må stemme over ens. Vi testet simpelthen ved å prøve å opprette en bruker med all tenkelig input; noen tomme felter, feil e-mail format, ulike passord og slikt. Denne måten å enhetsteste programmet på funket greit, men vi skjønte i ettertid at Junit-tester hadde vært en bedre løsning slik at vi ikke overså situasjoner som kunne oppstå og kaste exceptions.

Både den innebygde debuggeren i NetBeans og såkalt konsolltesting med System.out.println() ble brukt svært hyppig, kanskje spesielt i MySQLmetoder-klassen. Med disse to verktøyene ble arbeidet med å oppdage bugs svært effektivisert. Det var ofte problemer med at noen metoder i MySQLmetoder-klassen ikke fungerte slik vi ville – da brukte vi debuggeren til å se hvor mange linjer som ble kjørt av den buggete metoden, og fant raskt ut hvorfor metoden hadde uønsket oppførsel, og kunne dermed endre den.

9.1 Brukbarhetstester

Omtrent halveis uti utviklingsprosessen ble deler av systemet testet på noen få personer ved hjelp av en type uformell SUS-test. Ved hjelp av disse testene fikk vi rettet opp i grensesnitt-relaterte feil som ikke hadde blitt oppdaget eller prioritert av oss. Et eksempel på dette er at samtlige av testpersonene reagerte sterkt på grensesnittet i fanen «Legg til sau», der alle mente at det hele var veldig rotete og det var vanskelig å skjønne hva man måtte gjøre for å faktisk legge til en sau. Se figur 5.

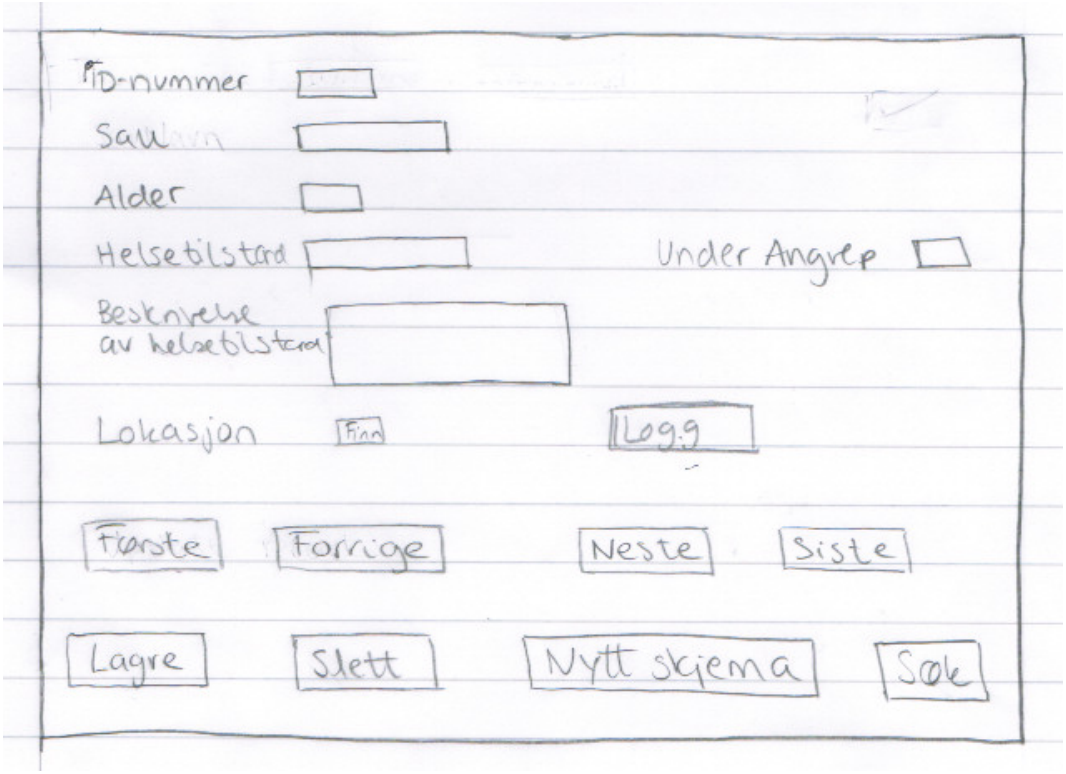
På bakgrunn av tilbakemeldingene vi fikk fra testpersonene, valgte vi å re-designe hele «Legg til sau»-fanen. Vi måtte gjøre det hele mye mer brukervennlig og intuitivt, og vi skjønte at denne fanen måtte ha så få felter som mulig og heller flytte «Søk» og «Logg» til egne faner. I tillegg ga det blant annet ikke mening å la brukeren opprette en sau og sette at den var under angrep ved opprettelse. I figur 6 vises det endelige designet av «Legg til sau»-fanen.

I tillegg var det noe forvirring rundt noen av feltene i «Registrering»-fanen, henholdsvis feltene for lengdegrad og breddegrad. Disse var i utgangspunktet helt normale tekstfelter, så testpersonene var ikke sikre på om man faktisk skulle skrive inn koordinater i tekstboksene. Dette ble endret ved å sette feltene til disabled (altså at de ikke kan editeres), og da blir brukeren nødt til å velge gårdsposisjon gjennom kartet (og dermed fylle inn bredde- og lengdegradsboksene) før en kan klikke på «Register»-knappen.

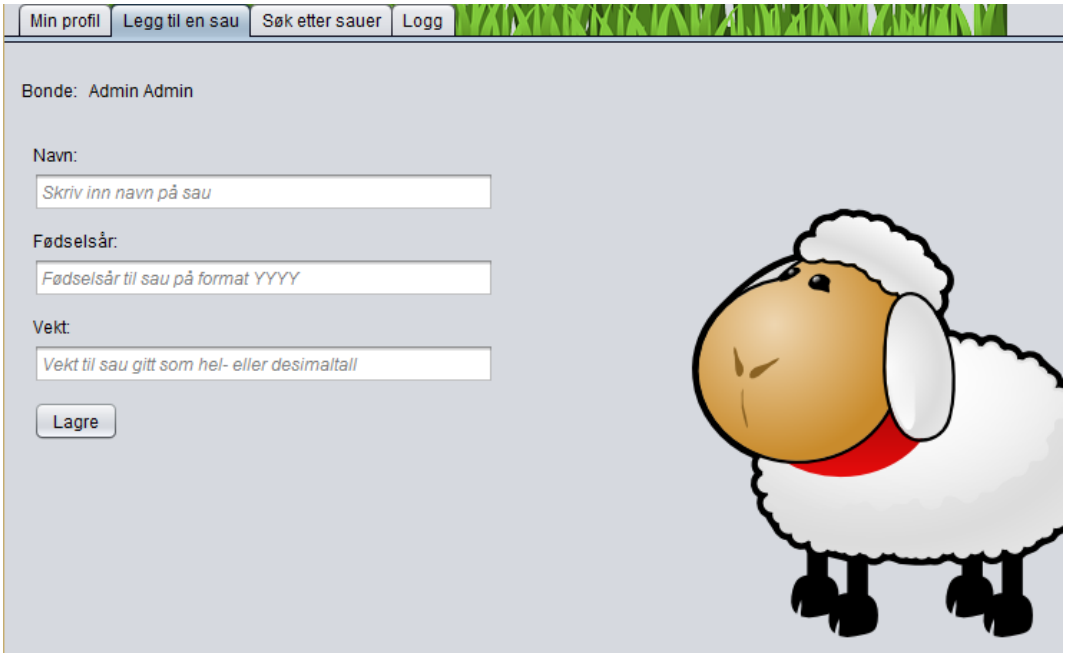
9.2 Systemtester

Se vedlegg 05-Systemtesting for detaljerte beskrivelser av hver enkelt test.

Systemtesting er en svært sentral del av programvareutvikling. I vårt tilfelle sikrer testene at alle kravspesifikasjoner er møtt med god margin og at alle moduler av applikasjonen fungerer som forventet ved alle former for input. Testene er laget hovedsakelig for å hindre at uventede situasjoner som svekker funksjonaliteten skal oppstå. Programmet skal kunne håndtere alt som brukeren kan finne på å sende inn, og tilbakemelding på om input er gyldig eller ikke – derfor er det svært viktig med gjennomgående testing. Eksempler



Figur 5: Første utkast til designet av fanen «Legg til sau».



Figur 6: Endelig design av fanen «Legg til sau».

på feilsituasjoner som kan oppstå som testene er designet til å fjerne, er ugyldig input i felt, feil validering av for eksempel dato eller backend-funksjonalitet som slutter å fungere.

To av de mest sentrale kravspesifikasjonene omhandler skalerbarhet og dimensjonering. Kravene er at systemet skal kunne håndtere minimum 200 bønder samtidig og minimum 10 000 sauer i systemet. Dette ble testet grundig ved å populere databasen med 200 bønder med 50 sauer hver registrert i sitt navn. Etter nøye tester ble det konkludert med at systemet fremdeles var like responsivt som da databasen ikke hadde noen sauer og kun én bruker registrert. Dette var overraskende, siden vi regnet med at responstiden for diverse funksjoner ville bli svært tregt. Vi prøvde også å registrere 10 000 sauer under én bruker, og hadde da samme resultat – alt fungerte svært responsivt. Eneste unntaket var at det tok litt lang tid å vise alle sauene på kartet, men dette er et uviktig punkt siden ingen bønder kommer til å ha 10 000 sauer i sitt navn samtidig.

Andre tester som responstid, tilgjengelighet og oppetid er dokumentert i vedlegg 05-Systemtesting.

10 Konklusjon

Nå som prosjektet er ferdig, produktet fungerer og rapporten er ferdigskrevet kan alle medlemmene i gruppe 12 endelig puste lettet ut. Et semester med IT1901 – Informatikk prosjektarbeid I, har gitt oss et innsikt i en annen tilnærming til hvordan man kan jobbe med informasjonsteknologi, og hvordan man kan implementere strukturerte arbeidsmetoder i et gruppeprosjekt. Ved første øyekast så Scrum ut som et stort tiltak rundt en relativt enkel programmeringsoppgave, men når vi er med veis ende er vi overrasket over hvor enkelt et par dokumentasjoner og flere oppdateringsmøter kan gjøre flyten og samarbeidet mellom gruppen og arbeidsmengden til å gå så lett. Vi har lært oss en arbeidsmetode som gjør kommunikasjon og synkronisering enkelt og oversiktlig. Vi har også lært nytten av forskjellige verktøy i planleggings- og utviklingsfasen av et system. Samtidig har vi fått en anelse av hvordan det er å jobbe med slike prosjekter i et selskap eller for en faktisk kunde, via teorien rundt Scrum som et eksempel på arbeidsmetode.

Vi har møtt på utfordringer både med tanke på prosessen og det faktiske programmet med programmering. Samtidig som vi har lært å jobbe sammen og strukturert, og å reflektere over vår egen arbeidsstruktur og -metode, har vi fått en strålende anledning til å sette oss inn i aspekter ved programmering som vi kanskje ikke fått utforsket utenom emnet. Vi har lært mye av den tekniske delen av prosjektet, noe vi antageligvis kan få god nytte av videre i lignende prosjekter.

Med det kan vi si at dette prosjektet har gitt oss en verdifull innsikt og lærdom rundt prosess, samarbeid og strukturering, og arkitektur- og systemutvikling.

11 Kilder

Scrum:

01.09.13 https://www.scrum.org/Scrum-Guides
01.09.13 http://agileatlas.org/atlas/scrum

Steg for å implementere Scrum:

05.09.13
http://www.allaboutagile.com/category/how-to-implement-scrum-in-10-easy-steps/

Burndown chart:

03.09.13 http://en.wikipedia.org/wiki/Burn_down_chart
06.09.13 http://apps.vanpuffelen.net/charts/burndown.jsp

Sekvensdiagram:

18.09.13 http://www.ibm.com/developerworks/rational/library/3101.html