Hovedrapport

Norges Teknisk-Naturvitenskapelige Universitet
21.11.2013

Informatikk Prosjektarbeid 1

Gruppe 19 - IT1901 - 2013

Fredrik Tørnvall

Hanne Marie Trelease

Jim Tørlen

Said Turusbekov

Edgar Vedvik

SHEEPOLINI

INNHOLD

1. INTRODUKSJON

1.1 OM PROSJEKTET OG FAGET

Dette er en rapport for et prosjekt gjennomført i faget Informatikk Prosjektarbeid 1(emnekode IT1901), ved Norges Teknisk-Naturvitenskapelige Universitet i Trondheim høsten 2013. Prosjektet og rapporten er utført av en gruppe på 5 studenter, som gjennomførte et mellomstort programmeringsprosjekt.

Det mest sentrale i faget er å forstå forholdet mellom produkt- og prosessorienterte utfordringer i programmeringsprosjekter. Her vil kunnskap om metode og teori for organisering av grupper i programmeringsprosjekter være sentralt. Målet med faget er å tilegne seg kunnskap om metoder og teorier for organisering av gruppeprogrammeringsprosjekter. Sentralt står forståelse for samspillet mellom prosess og produkt-orienterte utfordringer og aktiviteter i et programmeringsprosjekt.

Parallelt med denne rapporten vil hvert gruppemedlem ha utformet en kort individuell rapport som baserer seg på egne refleksjoner rundt hvordan prosjektet har vært. Her skal erfaringer og tanker rundt det å jobbe i gruppen presenteres, med en del fokus på egen erfaring og konkrete eksempler på hva man har lært.

1.2 OPPGAVEN

I oppgaveteksten heter det at vi er programutviklere i et programvareselskap som blir kontaktet av en sauebonde(kunden) som ønsker hjelp til å utvikle et system for registrering og administrering av sauene sine. Oppgaven er å komme med forslag til en løsning, for så å utvikle et system basert på dette.

Et av kriteriene for programmet går ut på at bonden skal kunne holde seg oppdatert på sauenes posisjon via kart, samt å loggføre tidligere posisjoner. Et annet er at programmet skal kunne ta imot alarmer fra sauene hvis de er under angrep eller har blitt drept, for så å videreformidle en melding til bonden. Alle kriteriene er listet opp i Kravspesifikasjonene(vedlegg nr. 07 *"Kravspesifikasjoner"*). Det vil gjennom prosjektet være hyppige kundemøter hvor progresjonen presenteres for kunden. Under disse kundemøtene vil en studieassistent i faget simulere kunden, men samtidig også være en veileder for gruppen.

1.3 OM GRUPPEN

Gruppen består av fem studenter som alle studerer til Bachelor i Informatikk ved NTNU. Disse er Fredrik Tørnvall, Hanne Marie Trelease, Jim Tørlen, Said Turusbekov og Edgar Vedvik. For mer informasjon om gruppemedlemmene, gå til 3.2.

1.4 VALG AV PROSESSMETODE

Vi valgte å løse oppgaven ved å lage en Android applikasjon. Dette fordi vi ønsket å lage et brukervennlig program som er lett tilgjengelig uansett hvor man er. Hovedgrunnen til at vi valgte Android som plattform, er at bonden lettere skal kunne få tilgang til tjenesten ved å logge seg på applikasjonen gjennom en mobil plattform. Kunden var positiv til valget av plattform. Den største utfordringen var riktignok at ingen i gruppen hadde tidligere erfaring med Android, noe som økte risikofaktoren rundt utviklingen.

2. TIDSESTIMERING

For å estimere hvor lang tid vi kom til å trenge på å utføre hver oppgave, og derpå hver sprint, har hele veien vært gjennom bruk av Planleggingspoker(se siste avsnitt i punkt 4.2). I de første sprintene var dette alt vi gjorde når det kommer til å estimere tidsbruk. Vi "spilte" planleggingspoker, satt inn tallene i sprintbackloggen(se vedlegg nr. 03 *"Sprinter"*), for så å gå gjennom hvor mye tid vi faktisk hadde brukt i slutten av sprinten.

Et problem vi oppdaget underveis var at vi ikke hadde god nok kontroll på progresjonen underveis i sprintene. Vi bestemte oss for at vi trengte et system for å overvåke dette på en daglig basis. Løsningen vi valgte å gå for, var å produsere et Burndown Chart i Excel, hvor vi for hver dag estimerte hvor mange timer som gjensto per oppgave. På denne måten fikk vi bedre oversikt over hvordan vi lå an til et hvert tidspunkt, og man kunne lettere se hvem som kanskje trengte hjelp med sine oppgaver. I tillegg førte vi en graf som viste hvor langt vi var kommet, i henhold til hva som gjensto.

Se vedlegg nr. 06 "*Product Backlog & Burndown Graf*" for beskrivelse av burndown med burndown-graf, og nr. 10 "*Burndown Chart*" for tabell med burndown-graf.

For å få oversikt over hva som måtte gjøres i løpet av utviklingsprosessen startet vi med å lage en arbeidsnedbrytningsstruktur, eller work breakdown structure(se vedlegg nr. 08 *"Skisser og Diagrammer"*) som det også heter. I denne strukturen organiserte vi gjøremålene i pakker ut fra hvilken kategori de forskjellige gjøremålene hørte til i. Vi satte også opp et GANTT-diagram(se vedlegg nr. 08 *"Skisser og Diagrammer"*) for å få omtrentlig oversikt over når de forskjellige pakkene i WBS-diagrammet kom til å bli ferdig, og når de kom til å være aktuelle i prosessen. Vi har også satt opp milepæler for å måle progresjon i diagrammet, ettersom det er lettere å forholde seg til mål som ligger nærmere i tid. Vi har stort sett satt systemfunksjoner og kravspesifikasjoner som milepæler, som har vært motiverende å forholde seg til underveis. GANTT-diagrammet viser når de forskjellige pakkene i WBS-diagrammet har foregått over tid og er delt opp i de fem sprintene vi har hatt.

Totalt på hele prosjektet, har vi estimert 270 timers arbeid, hvor 256 av dem falt under sprintene. Den siste biten på 14 timer, ble brukt til å forberede en demonstrasjon/presentasjon og å ferdigstille rapporten.

3. PROSJEKTORGANISERING

3.1 ANSVARSFORDELING

Ansvarsfordelingen i gruppen har vært fordelt etter som nye oppgaver har oppstått. Ansvar for ulike deler av hver sprint er også fordelt, og detaljene ligger i backlogene for hver sprint.

Se vedlegg nr. 03 "*Sprinter*" for sprint-backlog.

3.2 MEDLEMMENES KUNNSKAPSOMRÅDER

Medlemmene av prosjektgruppa sitter på forskjellige kunnskapsområder, noe som har gjort at vi utfyller hverandre godt som et helhetlig utviklerteam. Gruppa består av studenter som studerer informatikk ved NTNU.

3.2.1 FREDRIK TØRNVALL

Fredrik har erfaring med Python fra faget "IT grunnkurs", og i faget "objektorientert programmering" lå hovedvekten på Java. Begge fagene er fra førsteåret på informatikk ved NTNU. Dette har kommet godt med siden vi arbeidet en god del med Java i utviklingen. Han har også jobbet en del i team tidligere, blant annet på andre typer prosjekter i offentlig og privat sektor. Til eksempel var han leder med organisatorisk ansvar for et team på syv i forbindelse med filmfestivalen Kosmorama i Trondheim i 2010.

3.2.2 HANNE MARIE TRELEASE

Hanne Marie har tidligere studert på Universitetet i Oslo(UiO), hvor hun hadde faget "systemutvikling". I faget lærte hun en del teori rundt systemutvikling, scrum og forskjellige scrumverktøy, og har dermed erfaring med å lage diverse diagrammer. Diagrammene er nyttige for å holde kontroll over strukturen på det organisatoriske i programmet og utviklingsprosessen.

Hun har i tillegg erfaring med bruk av java og javadoc fra UiO, der hun tok fagene "grunnkurs i objektorientert programmering" og "objektorientert programmering" (begge var basert på java).

3.2.3 JIM TØRLEN

Jim har tidligere gjennomført "IT grunnkurs" med Python på Bachelor i Informatikk på NTNU. Hans sterkeste side er Java, som han brukte i faget "objektorientert programmering". Han har i tillegg tidligere erfaring med flere av verktøyene som ble brukt under prosjektet

Jim har god erfaring med å jobbe i team fra før. Eksempler på dette er vervet i bedriftskomiteen i linjeforeningen Online og vaktkommandørposten han hadde i forsvaret.

3.2.4 SAID TURUSBEKOV

Said har erfaring i "objektorientert programmering"(C++) og "IT grunnkurs"(Python) fra første året på Bachelor i Informatikk på NTNU.

Han har også drevet litt med HTML, og har opparbeidet seg tegneferdigheter over flere år selvstudie av Photoshop, Illustrator, generelt design, portrettegning, og lignende. Til slutt kan det nevnes at han har noe erfaring med iOS-utvikling.

3.2.5 EDGAR VEDVIK

Edgar kan Python og Java fra første året på informatikk, etter å ha hatt fagene "objektorientert programmering" og "IT grunnkurs". Han har også diverse erfaring fra frivillige verv. Før og under prosjektarbeidet var han medlem av drifts- og utviklingskomiteen(dotKom) i Linjeforeningen Online og IT-komiteen(ITK) på Studentersamfundet. Her har han fått mer erfaring med å arbeide i team, og en god del Python og PostgreSQL. Han har også erfaring med å jobbe i team fra tidligere verv. Edgar har også erfaring med servere, spesielt Debian, fra både ITK og dotKom.

3.3 ANSVARSOMRÅDER

Vi tok tidlig i prosessen en kartlegging av styrkene til gruppemedlemmene. Resultatet av dette ble at vi fordelte ansvarsområder på bakgrunn av hva gruppemedlemmene selv mente de var flinke til. Selv om ikke alle på gruppen hadde kunnskaper om det samme, valgte vi likevel å gi alle muligheten til å prøve seg på forskjellige deler av prosjektet. Dette for at alle skulle få innsikt i ulike deler av prosessen. Når det kommer til prosjektstyring og oppgaver i henhold til Scrum-metoden har vi samarbeidet for å løse disse oppgavene. Eksempler på oppgaver vi har samarbeidet om, er:

- Sprinter
- Kravspesifikasjoner
- Tidsestimat
- Versjonskontroll (Github)
- Testing

Utover fellesoppgavene har vi delt inn i ansvarsområder for hvert enkelt medlem av gruppen. Ansvarsområdene er fordelt etter hvilke egenskaper og kunnskaper de ulike gruppemedlemmene sitter på, dette for å optimalisere fremgangen og kvaliteten i arbeidet. De største ansvarsområdene hvert medlem har hatt, er:

FREDRIK Programmering Scrum-master	**HANNE MARIE** Programmering Diagrammer	**SAID** Design Programmering
EDGAR Programmering Database Server	**JIM** Rapport Programmering	

De forskjellige rollene ble inndelt i henhold til kunnskapsområdene. Vi diskuterte på første møtet de forskjellige rollene i Scrum og andre ansvarsområder. Det ble tydelig at erfaring fra tidligere prosjektarbeid kunne brukes til vår fordel i dette prosjektet.

Scrum-master ble valgt på bakgrunn av tidligere ledererfaring. Ansvar for server og database er ikke en del av Scrum, men vi fant det likevel nødvendig å gi en person hovedansvar for dette. Det ble derfor naturlig å gi ansvaret til personen med tidligere erfaring på det området. Vi hadde også en på gruppen som hadde erfaring med grafisk design, som nevnt tidligere. På samme måte ble det naturlig å tildele denne personen det grafiske ansvaret på prosjektet.

Kort oppsummert kan man si at rolleinndelingen er basert på hvilke kunnskaper de ulike medlemmene satt på før prosjektstart. Det skal likevel nevnes at ansvarsfordelingen ble tatt på møtene slik at medlemmene fikk muligheten til å ytre sine meninger.

4. PROSESSBESKRIVELSE

Vi vil nå se litt nærmere på hvordan fremgangsmåte vi har valgt, og de ulike aspektene i prosessen med å utvikle systemet.

4.1 SCRUM

Scrum er en arbeidsmetode som brukes for å organisere programvareutvikling. Scrum faller inn sammen med smidige systemutviklingsmodeller, noe som betyr at prosessen er iterativ og inkrementell. Dette vil si at fremdriften i prosjektet evalueres ut fra delmål og at utviklingen er modulbasert.

I Scrum vil prosjektet bli delt opp i mindre oppgaver og samlet i en felles "backlog". Se vedlegg nr. 06("*Product Backlog & Burndown Chart*") for vårt eksempel. Backlogen inneholder alle oppgavene som må gjøres for at prosjektet skal fullføres. Ettersom backlogen representerer hele prosjektet, er vi nødt til å dele den inn i mindre oppgaver. I Scrum kalles disse mindre delmålene for "sprinter". Sprintlengden fastsettes i starten av prosjektet, og alle sprintene skal

være like lange gjennom hele prosjektperioden.

Inndelingen av oppgaver for hver sprint skjer i starten av en sprint, hvor man har et sprintmøte. Målet for en sprint er å ta oppgaver fra backloggen og gjøre selve ideene om til fungerende kode. På denne måten får man en sprintbacklog som består av oppgaver som skal fullføres i den angitte perioden. I løpet av en sprint skal utviklergruppa, Scrum-master og produkteier møtes daglig til statusoppdateringer hvor man snakker om hva den enkelte har jobbet på og hva man skal jobbe med til neste statusmøte.

Når sprinten skal avsluttes har man en lenger versjon av dette møte hvor en presentasjon av den nye funksjonaliteten til produkteieren står sentralt. Her får produkteieren mulighet til å komme med tilbakemelding som kan føre til at utviklergruppa må gjøre endringer eller legge til nye oppgaver i backlogen. Til slutt i sprinten møtes alle til et refleksjonsmøte hvor man evaluerer prosessen og identifiserer forbedringspotensialet.

Scrum-metoden forholder seg til tre hovedroller i et prosjekt. De tre rollene som er definert i Scrum er; produkteier, utviklergruppe og Scrum-master. Produkteierens oppgave er å være kundens stemme, sørge for at gruppa skriver hensiktsmessige oppgaver inn i backlogen og at oppgavene blir prioritert riktig. Utviklergruppas oppgaver består av å produsere et leverbart produkt i løpet av hver sprint – slik at det kan demonstreres for kunden. På den måten får kunden hele tiden innsikt i status på prosjektet. Oppgaven til Scrum-masteren er å sørge for at utviklergruppa har de beste forutsettingene for å fullføre oppgavene sine, sørge for at retningslinjene til Scrum-metoden etterfølges, at møtene blir gjennomført, og å realisere gruppens forbedringspotensial.

4.2 VÅR BRUK AV SCRUM-METODEN

Scrum som organisasjonsverktøy for gruppa har naturlig nok begrenset seg noe med tanke på situasjonen vi er i. Ingen i gruppa hadde tidligere erfaring med Scrum og vi måtte derfor bruke litt tid i starten av prosjektet på å tilegne oss den kunnskapen vi trengte.

Vi har etterstrebet å følge metodikken så langt det har latt seg gjøre. Noen av endringene vi har gjort går blant annet på begrensning i møteaktivitet. Scrum krever, som nevnt tidligere, at gruppa med Scrum-master og prosjekteier møtes en gang daglig. Her valgte vi å begrense statusmøtene(se vedlegg nr. 04 *"Møtereferat og Kundemøte"*) til to ganger i uken (tirsdag og torsdag) samt et kundemøte i hver sprint. Lengden på sprintene har vi satt til to uker for å få nok tid til å ferdigstille modulene til kundemøtene.

Etter statusmøtene har vi ofte hatt lengre arbeidsøkter, som har vært på 3-5 timer. I løpet av øktene har vi jobbet sammen om problemer, hjulpet hverandre med oppgaver og drevet med parprogrammering. På denne måten fikk vi et mye bedre samarbeid i gruppa og vi hadde bedre tilgang på hjelp dersom det var noe vi ikke fikk til.

Tidsestimering(se punkt 2) og prioritering av oppgaver er også en viktig del av Scrum-metoden. Disse oppgavene har vi løst ved å se på hva som må gjøres for at neste modul skal være funksjonell til slutten av sprinten og prioritert de oppgavene høyest.

Tidsestimeringen løste vi ved å bruke "*planleggingspoker*", som er et kortspill man kan bruke i

Scrum for at alle i gruppa skal få muligheten til å komme med sine synspunkter på hvor lang tid hver oppgave vil ta. Her må gruppemedlemmene med høyest og lavest estimat argumentere for valget de har gjort. Basert på argumentene diskuterte vi i fellesskap forslagene, for så bestemme et timeantall som man felles var enige i.

4.3 PRODUKTBACKLOG

Backlog er en viktig del av Scrum. Vi valgte å se på kravspesifikasjonene, for så å formulere disse om til "user stories". User stories er til for å kartlegge funksjonaliteten som må være med i sluttproduktet. Disse har vi delt inn slik at hver story representerer en funksjonalitet som er i kravspesifikasjonene(se vedlegg nr. 07 *"Kravspesifikasjoner"*). De punktene i kravspesifikasjonen som gikk ut på responstid, kapasitet ol. tolket vi som ikke-funksjonelle krav og tok ikke med disse som user stories/use cases. Vi tok for oss én user story av gangen, og satt dem inn i hver sin sprint. Deretter delte vi user stories videre inn i mindre oppgaver som måtte gjøres for at en problemstilling skulle være løst. Et eksempel på dette er; "Som bruker vil jeg kunne logge meg inn". Alle oppgavene som utgjør denne storien blir satt i samme sprint, og definerer arbeidsoppgavene for sprinten.

Backlogen brukte vi videre som utgangspunkt når vi skulle sette sammen sprintene. Vi valgte å sette sammen en sprint av gangen for så å ha et avslutningsmøte på slutten av hver sprint. Her evaluerte vi hvordan arbeidet hadde fungert på den avsluttende sprinten. Sentrale problemstillinger vi tok opp for hver sprintavslutning var henholdsvis:

- Hva var bra?
- Hva var dårlig?
- Hva skal vi gjøre for å forbedre oss til neste gang?

Etter evaluerings-runden, og litt diskusjon rundt sprinten, starter arbeidet med å formulere neste sprint. Måten vi gjorde dette på, var å se på neste user story i backloggen for så å finne oppgaver som var knyttet til denne. Estimering av tid ble gjort på samme måte i alle sprintene og tildeling av oppgaver skjedde dynamisk. Gruppemedlemmene valgte seg ut de oppgavene som de selv hadde motivasjon for å gjennomføre.

Målet til gruppa har vært at en sprint skal inneholde en funksjonalitet, slik at man i slutten av sprinten skal kunne teste koden og vise resultatet til kunden. Lengden på en spirt har vært satt til to uker. I en sprint får gruppemedlemmene tildelt arbeidsoppgaver som de har ansvaret for at skal bli ferdig, og hvis ikke alle oppgavene blir 100 % ferdige innen tidsfristen flyttes den til neste sprint.

4.4 SPRINT INNDELING

4.4.1 SPRINT 1

User-Story: *"Som bonde ønsker jeg å kunne logge inn på en profil med brukernavn og passord".*

For at denne delen av kravspesifikasjonene skulle virke var den en god del som måtte settes opp i starten av prosjektet. Vi måtte derfor se på alle elementene som trengtes for at brukeren skulle

kunne gjøre det han/hun ønsket. I tillegg til systemkravene var det også krav til dokumentasjon, slik at vi skulle få en oversikt over hvordan systemet skulle designes.

Vi startet derfor med å sette opp et Use-case diagram(se vedlegg nr. 08*"Skisser og Diagrammer"*) som beskriver hvordan de forskjellige delene av systemet er avhengig av hverandre. Dette er med på å gi en oversikt over hva en bonde skal kunne gjøre med programmet. Deretter lagde vi et aktivitetsdiagram(se vedlegg nr. 08*"Skisser og Diagrammer"*) for å planlegge hvordan flyten i programmet skulle bli. Dette ga oss et utgangspunkt å forholde oss til når vi skulle programmere de forskjellige klassene.

Vi tok en innledende risikoanalyse og opprettet en risikotabell for å få oversikt over hvilke faktorer som kunne påvirke prosjektet negativt. En av de store utfordringene vi identifiserte var mangelen på kunnskap. Ingen i gruppa hadde jobbet med Scrum eller et så stort programmeringsprosjekt. Diagrammene vi produserte i Sprint 1 var viktig for prosjektstyringen, og for at alle gruppemedlemmene skulle få en oversikt over hvor mye arbeid som var påkrevd før fristene.

Utover dette var det mer enn bare planlegging og diagramproduksjon vi fikk gjort i første sprint. Vi innså tidlig at programmeringsmengden kom til å bli stor på den korte tiden vi hadde, og valgte derfor å sette opp en server med database tidlig. Det denne skulle være i stand til etter første sprint, var å lagre brukerinformasjon. Vi skulle i tillegg kode en "login-klasse" på front-end. Målet vårt var at vi i slutten av sprinten skulle kunne demonstrere at pålogging med validering av brukerinformasjon på serversiden var ferdigstilt.

Som man ser fra sprintbackloggen(vedlegg nr. 03 "*Sprinter*") er det ikke alle elementene som har fått status som 100 % ferdig. Dette har mye med at vi ikke hadde noen erfaring med Android som plattform fra før, noe som ga oss noen store utfordringer. Et godt eksempel, er utfordringen vi hadde med å lage et klassediagram(se vedlegg nr. 08*"Skisser og Diagrammer"*). Årsaken til at vi lagde dette, var å få en oversikt over systemets struktur. Først etter at vi hadde fått bedre oversikt over hvordan plattformen fungerte, måtte vi revidere klassediagrammet for at det skulle være realistisk i forhold til den plattformen vi jobber på.

4.4.2 SPRINT 2

User-Story: *"Som bonde ønsker jeg å kunne registrere opplysninger om mine sauer med id, navn, alder, vekt, helseinformasjon og geografisk lokasjon om en enkelt sau".*

I denne sprinten måtte vi få registrering av brukere og sauer til å virke. Vi ble ferdige med de fleste av oppgavene i sprinten, med noen få unntak. Som vist i backloggen(vedlegg nr. 03 "*Sprinter*") fra sprint 2 kan vi se at "Kode registrering av sauer" kun er 70 % fullført. Grunnen til det er at kommunikasjonen mellom Android applikasjonen og serveren ikke ble ferdigstilt, noe som igjen var et resultat at de som skulle samarbeide om dette ikke hadde fått tilstrekkelig med tid.

Klassediagrammet og testklasse ble heller ikke helt ferdige. Førstnevnte jobbet vi videre med fra forrige sprint for å forbedre det i henhold til hvordan applikasjonen skulle se ut. Men selv om vi brukte litt tid på det, ble diagrammet ikke ferdigstilt da vi fortsatt ikke hadde god nok oversikt over hvordan aktivitetene i en Android applikasjon hang sammen. Vi kan derfor konkludere

med at vi ikke nådde målsettingen vår for sprint 2 og måtte flytte noen av oppgavene videre til sprint 3. Hovedårsaken til dette, var at vi forsøkte å gjøre for mange og tidkrevende oppgaver på én sprint.

4.4.3 SPRINT 3

User-Story: *"Som bonde ønsker jeg å kunne slå opp i systemet og redigere informasjon om en sau og om meg selv".*

Denne sprinten markerer at vi er halvveis i prosjektet, og at applikasjonen begynner å ta form. I backloggen(vedlegg nr. 03"*Sprinter*") ser vi at oppgavene begynner å bli bedre definert for å begrense størrelsen på arbeidsmengden. Vi kan også se i backloggen at vi fullførte klassediagrammet, og at oppgavene knyttet til rapporten er bedre spesifisert slik det blir lettere å evaluere fremdriften. I denne sprinten endte vi med å fullføre alle oppgavene 100 %, noe som gjenspeiler at vi har tatt med oss erfaringer fra forrige sprint. Vi kan med trygghet si at vi traff målsettingen denne gangen.

I løpet av denne sprinten hadde vi også en midtveispresentasjon hvor vi forklarte prosessen så langt. Her så vi også på hvilke verktøy vi hadde brukt, hvorfor vi hadde valgt dem, og hvorfor vi hadde valgt å gå bort fra noen verktøy vi hadde brukt tidlig i prosessen.

4.4.4 SPRINT 4

User-Story: *"Som bonde ønsker jeg å se på et kart hvor en sau sist var registrert. Som en bonde ønsker jeg å kunne se på tidligere logger".*

I denne sprinten måtte vi inkludere to user-stories for bli ferdig til tiden. Dette medførte at det ble unormalt mange oppgaver i sprinten. I tillegg til programmering ble det også nødvendig å se på arkitekturen og designet til applikasjonen. Med tanke på hvor mange oppgaver som ble inkludert i denne sprinten kom vi godt ut av det med bare en oppgave som fikk status "80 % ferdig". Her viste filtrering og søk seg å være en mye mer tidkrevende oppgave en først antatt. Etter å ha brukt mer tid enn estimert, måtte vi avslutte sprinten og flytte den gjenstående oppgaven til neste sprint. Målsetningen ble nesten møtt.

4.4.5 SPRINT 5

User-Story: *"Som bonde ønsker jeg å få beskjed/alarm når en sau er under angrep".*

Sprint 5 er den siste sprinten i prosjektet, og inkluderer bl.a. rapport og siste innspurt på koden. En av de største funksjonene som ble lagt til her, var alarmfunksjonen med automatiske oppdatering til applikasjonen. Dette inkluderer de 3 daglige oppdateringene av posisjoner, samt alarm ved angrep og alarm hvis sauen beveger seg utenfor bondens område.

Vi valgte også å løse problemstillingen rundt beiteområde ved og automatisk tildele bonden et område ved registrering av bruker. Grunnen til at løsningen ble gjort på denne måten var at vi ikke hadde tid til å lage en mer brukervennlig løsning. Vi vet at det er et forbedringspotensial, og hadde vi hatt tid ville en annen løsning på dette problemet vært å la bonden definere diagonaler

på området sitt med koordinater. Med andre ord, å gjøre det mulig for bonden å tegne opp området sitt på et kart i applikasjonen under registrering og endring av bruker.

4.5 RISIKOFAKTORER

Under en utviklingsprosess er det alltid risiko involvert. For å prøve å begrense sannsynligheten for at noe uheldig skulle inntreffe, gjorde vi en risikoanalyse i starten av prosjektet. Det forekommer ofte endringer underveis i et prosjekt som dette, noe som kan være en risiko i seg selv. Det er derfor nyttig å ha en plan for å håndtere slike utfordringer på en god måte.

Etter å ha tatt en risikoanalyse satte vi opp en risikotabell(vedlegg nr. 09*"Risikotabell"*). Tabellen viser sannsynligheten for at noe uheldig skulle kunne skje, og konsekvensen er satt på en skala fra 1-5, hvor fem er det verste utfallet. Tiltak for å unngå at en risiko skulle inntreffe, samt reparerende tiltak dersom risikoen faktisk skulle inntreffe ble og vurdert. Vi valgte å gi hele gruppa ansvaret med og kontinuerlig begrense risikoen for at problemer skulle oppstå.

Underveis i prosjektet har vi prøvd å forholde oss til risikoanalysen og følge de avvergende tiltakene for å unngå at noe skulle gå galt – dette med et ekstra fokus på de punktene med høy sannsynlighet og stor konsekvens. Tabellen er som nevnt utarbeidet helt i starten av prosjektet, og vi har revidert analysen et par ganger underveis. Da den originale tabellen stort sett har vært oppfyllende gjennom hele prosjektet, har det ikke vært nødvendig å gjøre store forandringer.

I tabellen står "manglende kompetanse" oppført som det mest kritiske punktet. Det er også på dette området av prosjektet vi har møtt de største utfordringene. Som nevnt hadde ingen av oss prøvd oss på Android-utvikling da vi startet prosjektet. Det har derfor krevd en del ekstra tid å sette seg inn i hvordan Android fungerer. Vi mener en del av utfordringene vi har hatt underveis har bidratt til å gjøre prosjektet utrolig spennende og lærerikt.

4.5.1 EKSEMPLER

Vi skal nå se nærmere på noen konkrete eksempler rundt utfordringene vi har hatt knyttet til de punkene vi har satt opp i risikotabell(se vedlegg nr. 09*"Risikotabell"*).

Mot slutten hadde vi en del problemer med "out of memory"-feil på enheter med mindre RAM enn test-tableten vår. Dette har vi ikke vært i stand til å rette opp i ettersom vi mangler erfaring og kunnskap med hvordan Android håndterer bilder og mellomlagring. Vi har måttet gi slipp på gode ideer fordi vi ikke har hatt tid til å tilegne oss den nødvendige kunnskapen for å realisere dem. Vi ønsket for eksempel å kunne krysse av for flere sauer samtidig i sauelisten, for så å vise de utvalgte på kartet. Dette ble for omfattende i denne omgang, men vi fikk lagt til muligheten å se på kun én sau på kartet av gangen.

I tillegg syntes vi at koden vår etter hvert ble for stor og rotete som følge av manglende struktur i begynnelsen. Mye av årsaken til dette slår igjen rot i at vi i starten manglet erfaring med utvikling i så stor skala, og med Android. Vi hadde rett og slett ikke innsikten i hvordan man går frem for å planlegge en kodeoppbygging. Dette medførte mye unødvendig kode som kunne blitt unngått.

En utfordring vi møtte på etter siste sprint, var at vi ikke fikk til å generere javadoc, som det står

spesifisert i disposisjonen at vi skal ha med. Selv etter å ha sittet flere timer og prøvd flere muligheter og spurt folk om hjelp, har vi ikke fått til å generere javadoc til html. Etter å ha lest litt på nettet om problemet ser vi at generering av javadoc ved bruk av Android kan være en omfattende prosess. Enten må man laste ned diverse plug-ins, eller så kan man bruke «juksemetoder» vi dessverre ikke forstod oss på. Igjen går denne utfordringen på manglende kompetanse, men vi har fortsatt tatt med javadoc-kommentarene i koden.

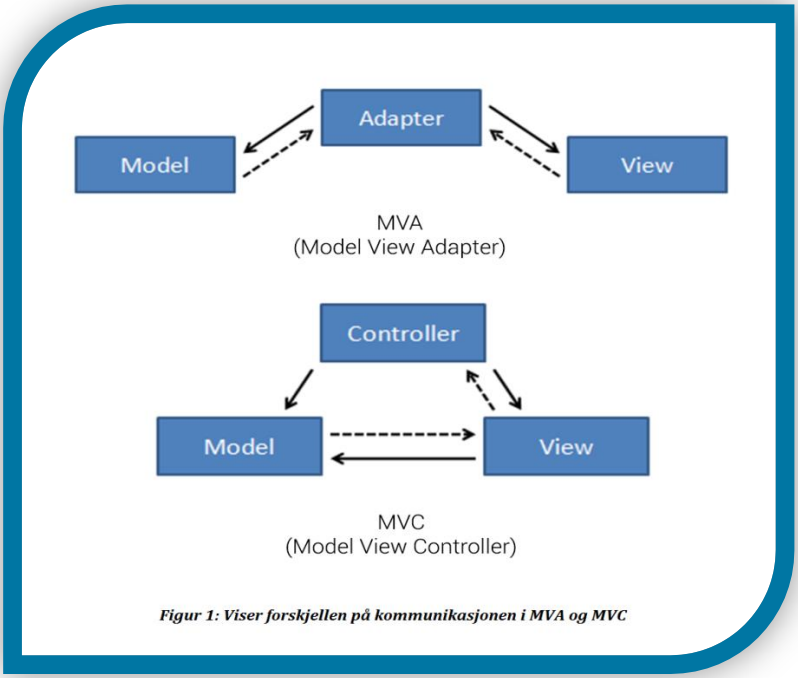
I forhold til punktene vi satt opp i risikoanalysen har det ikke vært mange problemer vi ikke har løst, og av de få vi har kommet over har de fleste vært kompetanserelaterte. Utover dette har det vært utfordringer rundt at vi ikke har overholdt avtaler om møtetider. Dette har vi underveis forbedret oss en god del på, men dersom vi hadde satt kortere møtetider fra start av, hadde mest sannsynlig flere kommet tidsnok. Vi valgte til tross for dette å fortsette med lengre arbeidsøkter som ga oss bedre mulighet til å jobbe sammen og til å hjelpe hverandre.

5. ARKITEKTUR FORKLARING OG SYSTEMFUNKSJONALITET

IT-arkitektur omhandler et sett av strukturer som er nødvendige for at et system skal kunne fungere sammen med eksterne komponenter, og egenskaper for de ulike komponentene. Vi vil her se nærmere på hvordan alle de ulike delene i systemet forholder seg til hverandre, og hva som knytter dem sammen.

For bedre å forstå arkitekturen til programmet denne rapporten omhandler, vil vi først se på hva "model view adapter" er.

5.1 MODEL-VIEW-ADAPTER(MVA) & MODEL-VIEW-CONTROLLER(MVC)



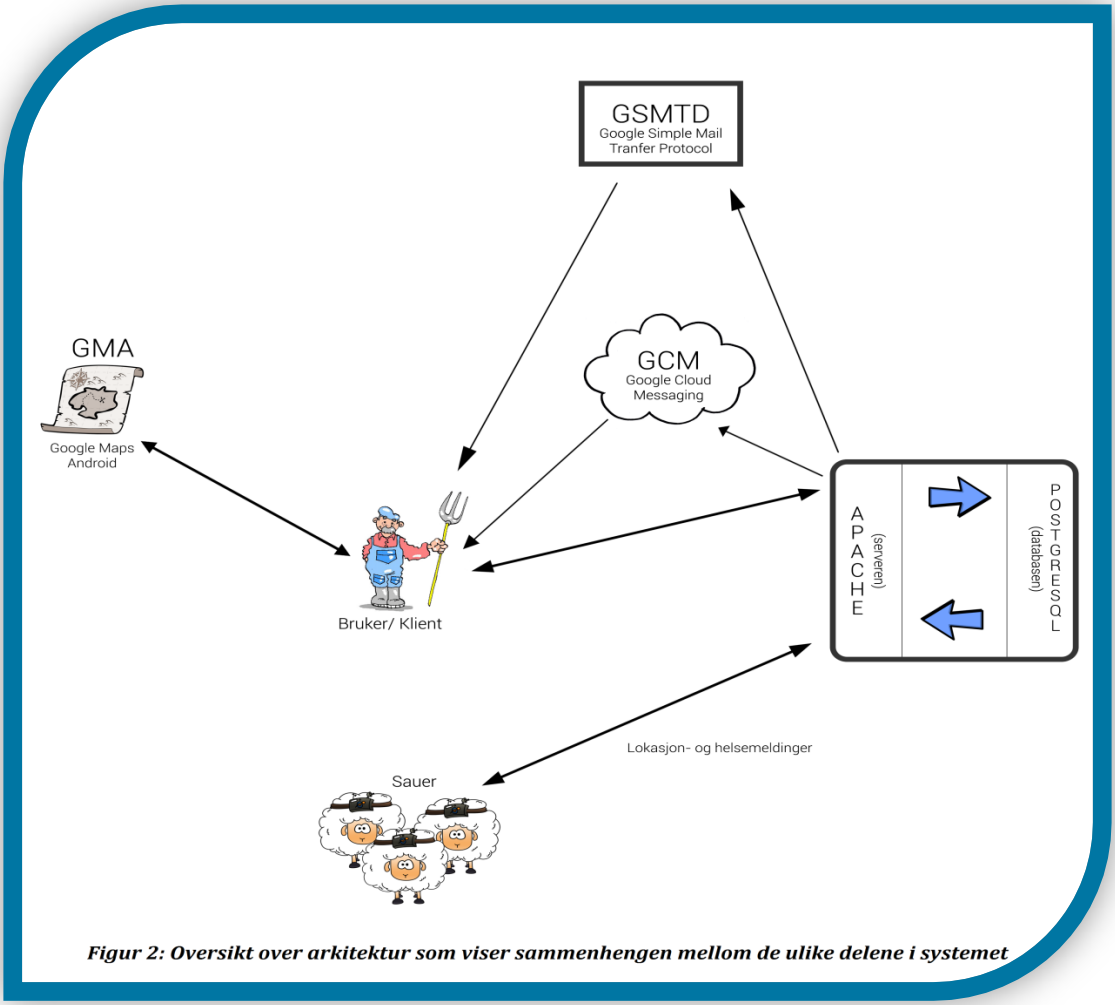
Figur 1: Viser forskjellen på kommunikasjonen i MVA og MVC

Model-view-adapter(MVA) er et designmønster man kan bruke i programvareutvikling. Designmønsteret stammer fra et annet som heter model-view-controller(MVC), og de har mange fellestrekk, men noen viktige forskjeller. MVC er veldig typisk innenfor webgrensesnitt, og MVC-programmer deler programmet opp i data, brukergrensesnitt og en kontrollenhet, henholdsvis "model", "view" og "controller". Controller er en komponent som ligger mellom model og view, og sørger for kommunikasjon.

De samme egenskapene har MVA, hvor adapter tilsvarer controller. Hovedforskjellen som gjør at dette programmet ikke er basert på MVC men MVA, er at vår model og view aldri kommuniserer direkte, uten å ha vært innom adapter(eller controller). Dette er ikke nødvendigvis tilfelle med MVC(se *figur 1*).

5.2 APPLIKASJONENS ARKITEKTUR OG SYSTEMFUNKSJONALITET

Som nevnt under punkt 5.1, har systemet et designmønster som ligner mer på MVA enn MVC. Kjernen er databasen(PostgreSQL), og inneholder all informasjon om brukerkontoer, sauelister, posisjonslogg, helseinformasjon, osv. Mer informasjon om PostgreSQL og årsak til at vi valgte dette finnes i verktøylisten(se vedlegg nr. 05 *"Verktøy"*).



Figur 2: Oversikt over arkitektur som viser sammenhengen mellom de ulike delene i systemet

Databasen vil i henhold til MVA-modellen(se *figur 1*) tilsvare "model", og brukergrensesnittet "view" i systemet. Gjennom brukergrensesnittet kan brukeren gjøre endringer i databasen. Denne kommunikasjonen skjer ved at brukergrensesnittet går via "adapter" – selve applikasjonen – for å komme gjennom til databasen. Med andre ord består klientprogrammet, altså applikasjonen vi har kalt Sheepolini, av et API som kommuniserer med databasen. Antall klienter som kan koble seg opp mot databasen og kjøre samtidig møter kravspesifikasjonene, som er på blant annet å kjøre for 200 brukere og 10 000 sauer samtidig(se vedlegg nr. 07 *"Kravspesifikasjoner"* for mer info*)*. Måten hele systemet kommuniserer på er omfattende, og vi vil ta for oss hvordan alle de ulike delene kommuniserer på. For en oversikt over arkitekturen, se *figur 2*(over).

Hvis man ønsker å opprette en ny bruker, eller ikke har en fra før, kan man enkelt gjøre dette via brukergrensesnittet, for så å logge på. Pålogging gjøres ved at applikasjonen i klientenheten kontakter databasen, via Raspberry Pi(RPI), for å sjekke at passord til brukeren stemmer med innloggingsinformasjonen. Videre vil informasjonen som ligger lokalt på enheten pares med informasjonen som ligger i databasen. Med paring menes det at enheten vil lagre en kopi av utvalgt informasjon knyttet til brukerkontoen liggende i databasen. Sistnevnte er satt til en uke for å begrense hvor mye ressurser systemet krever av de ulike komponentene.

Når man logger av slettes informasjonen i den lokale databasen på enheten. Når man så logger på igjen, vil den nyeste informasjonen liggende i RPI-databasen lagres lokalt igjen, med blant annet oppdaterte GPS-koordinater osv. På denne måten forsikrer vi oss om at posisjoner eldre enn en uke slettes fra klientenheten.

GPS posisjonene til sauene oppdateres hver åttende time, og gjøres ved at det sendes meldinger fra sauene til serveren. Disse meldingene inneholder koordinater og status for hver enkelt sau, og blir behandlet av serveren som setter den nye dataen inn i databasen(PostgreSQL). Kommunikasjonen mellom serveren og sauene er ikke implementert i systemet, da det ikke er en del av kravspesifikasjonene(se vedlegg nr. 07 *"Kravspesifikasjoner"*). Derfor har vi laget en simulator som kjøres på serveren 3 ganger i døgnet for å simulere hendelser og posisjonsendringer. I simulatoren er det lagt inn en sannsynlighet på 0,0005 % for at sauene kan dø eller bli angrepet, som tilsier at den totale sannsynligheten for at noe kritisk skal inntreffe er på 0,001 %.

Som nevnt oppdateres helsestatus og posisjonene til sauene hver åttende time, men vi har også valgt å legge til en ekstra funksjon som gjør at brukeren kan etterspørre oppdatering selv, når det måtte være ønskelig, via brukergrensesnittet. Hvis brukeren gjør dette, vil applikasjonen sende en forespørsel til RPI, som videre kaller på sauene(simulatoren) for å få nye posisjoner. RPI vil lagre de nye posisjonene i loggen for hver sau, for så å erstatte den gamle loggen i enheten med den nye. Når oppdateringene er ferdige, kan brukeren gå til kartet for å se hvor sauene er.

Kartfunksjonen vi har valgt å bruke er hentet fra Google Maps Android API(GMA API) som er en del av Google Services. For at brukeren skal kunne få tilgang til kartet, vil applikasjonen sende en forespørsel til GMA API sammen med en API-nøkkel. Dette gjøres for å få verifisert at klienten har tilgang til kartdataen. Før dette skjer vil et kartutsnitt defineres ved et sett angitte parametere, basert på posisjonene til sauene tilhørende brukeren. Utsnittet lastes så ned og blir liggende lokalt på enheten, og sauene tegnes opp ut fra posisjonene. Hver gang man zoomer inn

og ut, eller oppdaterer kartet vil applikasjonen sende en ny forespørsel til GMA API, da uten parameterne for saueposisjonene. For mer informasjon om GMA API, se verktøylisten(vedlegg nr. 05*"Verktøy"*).

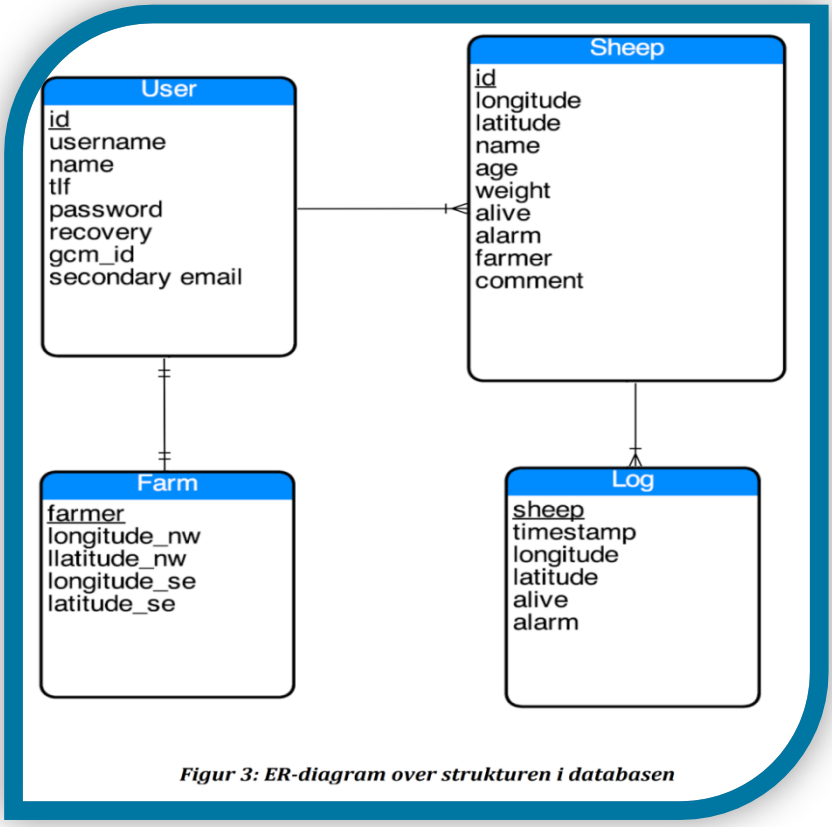
Hvis en sau skulle være død eller under angrep, vil RPI motta en melding om dette fra sauen. Serveren vil behandle informasjonen i meldingen, for så å sende ut to varslinger til brukeren. Den ene går til GCM(Google Cloud Messaging), som igjen varsler enheten slik at brukeren får en notifikasjon på enheten. Den andre varslingen sendes ut fra RPI og går til GSMTP(Google simple mail transfer protocol), som videresender statusen om den aktuelle sauen til brukerens registrerte E-mailadresser. Brukeren har mulighet til å ha opptil to E-mailadresser, hvor den ene er knyttet til påloggingsinformasjonen, og den andre kun til varslinger. E-mailadressen brukeren får varslingen fra er findmyherd@gmail.com. I tillegg til å sende varslinger til brukeren, vil statusinformasjon om sauene lagres i PostgreSQL-databasen, og logglistene på enheten erstattes av oppdaterte loggene fra RPI.

For mer informasjon om systemfunksjonalitet, se vedlegg nr. 02(*"Brukermanual(ikke-teknisk)"*).

5.3 RASPBERRY PI(RPI) OG DATABASE

Vi vil nå se litt nærmere på hvordan databasen er bygd opp og konfigurert, samt relasjonene mellom de ulike entitetene.

5.3.1 RELASJONER



Figur 3: ER-diagram over strukturen i databasen

Databasen vår består av fire entiteter(se *figur 3* for ER-diagram); "users", "farm", "sheep", og "log". Førstnevnte inneholder informasjon om alle bøndene som bruker applikasjonen. Her blir ting som kontakt- og innloggingsinformasjon lagret. Passordene er hashet med SHA1-kryptering, noe vi mener er tilstrekkelig for sikkerheten i applikasjonen. I tillegg til SHA1 kryptering har vi "krydret"(eng. salting) passordet med en tilfeldig generert 16-tegns lang streng. Dette er gjort for å unngå at en inntrenger skulle få tak i passordene ved bruk av tabeller med allerede genererte "hashes" for å knekke passordet til brukeren. Vi mener at dette er en sikker metode, fordi det i praksis fører til at det vil ta utrolig lang tid å knekke et passord. Dette til tross, skal nevnes at det er teoretisk mulig å knekke krypteringen, men det er ikke et kjent problem.

Til hver bonde er det koblet en gård("farm") som inneholder GPS-kordinater til beiteområdets hjørner. En bonde har også en eller flere sauer. Hver sau er tilknyttet en tabell hvor det er lagret helseinformasjon og GPS-data. Sauene har også to statusvariabler, henholdsvis “alive” og “alarm”. Dette gjør at vi raskt kan sjekke databasen om noe kritisk har skjedd.

Hver sau er koblet opp mot en logg, som inneholder tabellene med informasjonen, som vi nevnte tidligere. I loggen blir posisjon, status og tidspunkt for hver hendelse lagret. Hvis man for eksempel skulle lure på når en spesifikk sau sist var under angrep, kan man enkelt sende en spørring til loggen og få svar. For lettere å få tilgang til denne informasjonen, har vi valgt å lagre loggene i databasen i stedet for i en egen tekstfil. Dette fordi det fører til at databasen blir betydelig raskere til å gjøre spørringer. De tre ulike hendelsene som kan inntreffe sauene, er henholdsvis:

a. At sauen har beveget seg
b. At sauen har blitt angrepet
c. At sauen har dødd

5.3.2 TEKNISK/KONFIGURASJON

Siden vi ikke hadde tilgang på en ordentlig server, valgte vi å bruke en Raspberry PI til server under prosjektet. Den oppnår kravet for datalagringsmengde og responstid som var gitt i kravspesifikasjonen. RPI'en kjører Raspbian som er en gratis og uoffisiell utgave av Debian tilpasset RPI. Vi har valgt å benytte Apache 2.2 som http-server, og PostgreSQL som databasehåndteringssystem.

Apache håndterer innkommende HTTP-forespørsler og delegerer de videre til rett fil for så å kjøre filen som et Python skript. Skriptet returnerer en kode(f. eks 200) avhengig av status for handlingen. Noen skript returnerer også en JSON-formatert streng som inneholder data som skal bli sendt tilbake til applikasjonen. Python-skriptene får inn data fra HTTP-forespørslene og kommuniserer så med databasen før den returnerer eventuelle data.

PostgreSQL er et svært kraftig databasehåndteringssystem med åpen kildekode som fungerer utmerket på systemet til RPI’en vår. I tillegg til å ha PostgreSQL på serveren som hoved database, bruker vi en SQLite-database i selve applikasjonen som inneholder data for den gjeldende brukeren. Grunnen til at vi her bruker SQLite er fordi det er det som er enklest og best å bruke på Android. Lokalt lagrer vi informasjon om sauer, en ukes log og brukerinformasjon

bortsett fra passord.

For at applikasjonen skal holde seg oppdatert med den nyeste informasjonen bruker vi Google Cloud Messaging. Dette gjør at vi kan dytte oppdateringer på applikasjonen i bakgrunnen selv om applikasjonen ikke kjører. Hvis serveren vår får vite at en sau er under angrep eller har omkommet vil brukeren få en notifikasjon og epost om det umiddelbart gjennom Google Cloud Messaging.

6. TESTING OG DIMENSJONERING

6.1 TESTING

Under utviklingen av applikasjonen følte vi behovet for å ha automatiske tester for å sikre at utvikling av ny funksjonalitet ville fungere sammen med eldre funksjonalitet. Vi valgte å bruke Javas JUnit testrammeverk til testing.

En slik test innebærer at når vi sendte HTTP-forespørsler til Raspberry PI-en fra applikasjonen, ville forespørselen sjekkes, og returnere en statusmelding. Statusmeldingen ville være positiv hvis forespørselen innehold gyldig informasjon, mens hvis det var noe galt, for eksempel felter som manglet eller at man oppga feil passord, ville man motta en negativ statusmelding. Disse meldingene er representert med koder bestående av tre siffer avhengig av hva som gikk galt/riktig. Testklassene vi har laget til dette, ligger vedlagt i mappen som heter "JUnit". Fremgangsmåte for å finne disse:

Åpne zip-filen “Gruppe 19” og naviger slik:

Gruppe 19>kode.zip>it1901>JUnit

I tillegg til automatiske tester gjorde vi noen brukertester underveis. Dette vil si at vi gav en tablet med applikasjonen installert til en uvitende person som ikke har vært med på utviklingsprosessen. Hensikten med dette var å finne feil som vi ikke kunne funnet selv. En annen fordel med brukertestene var at vi fikk tilbakemeldinger på brukervennligheten. Dette gjorde at vi kunne endre en del detaljer som vi kanskje ellers ikke hadde tenkt på. Gyldigheten til brukertestingen var ikke helt optimal siden vi primært har testet applikasjonen på bekjente. Hadde vi derimot testet den på en bonde hadde vi kanskje funnet andre utfordringer rundt brukervennligheten.

Et eksempel på en brukertest som fikk oss til å gjøre endringer, er hvor vi så oss nødt til å flytte på knappene for å registrere bruker og gjenopprette passord. Begge disse knappene lå plassert slik at tastaturet dekket dem under pålogging.

6.2 DIMENSJONERING

Kravspesifikasjonen sier at systemet skal kunne greie å håndtere minst 200 bønder og 10.000 sauer. Vi har derfor dyttet masse testdata inn in databasen slik at hver bonde har minst 50 sauer. Systemet håndterer dette fint og fikk ingen utspill når det kommer til brukeropplevelsen.

En annen ting vi følte for å teste, var å gi én bonde et veldig høyt antall sauer. Selv ikke dette fikk store konsekvenser, og systemet håndterte også dette på en tilfredsstillende måte. Ettersom at det å gi brukeren mulighet til å tvinge oppdateringer ikke er en del av kravspesifikasjonene, men en ekstrafunksjon, valgte vi å si oss fornøyde med resultatet.

Om man skulle utvikle kravspesifikasjonene ytterligere, ville vi nok valgt å gå for en sterkere server enn en Raspberry PI. Vi visste allerede fra starten at en Raspberry PI ikke var egnet til å håndtere stort mer enn det kravspesifikasjonene sier.

7. SYSTEMDESIGN

Etter at vi bestemte oss for å lage en Android applikasjon ble vi enige, ganske tidlig i utviklingsfasen, at layout og design skal være enkelt å forstå for alle brukere uavhengig av forkunnskaper. Som hovedskjerm til programmet valgte vi å ha [kartvisning], slik at brukeren/bonden skulle få full oversikt over sauene sine med en gang applikasjonen starter. Her har vi også en meny-bar med mulighet for å skifte kartvisning, logge ut, slå opp liste med sauer og se egen profil. Tekstvinduer og knapper på skjermen valgte vi å ha middels store for å gjøre det enklere for brukeren å navigere i applikasjonen, og samtidig beholde mest mulig informasjon på skjermen.

Vi tegnet flere skisser(se vedlegg nr. 08 *"Skisser og Diagrammer"*) med ulike potensielle logoer. Å velge hvilken av skissene vi skule videreutvikle til en logo gjorde vi ved en avstemming. Bakgrunnsfargen til applikasjonen vår ble valgt til blå siden det er en avslappende farge. Knappene måtte passe til bakgrunnen, uten å tiltrekke for mye oppmerksomhet, men samtidig være godt synlig. Den fargen som passet best til dette ble oransje, da den er komplementærfargen til blå. Vi brukte den samme fargen på alle knappene bortsett fra "Slett"-knappen som er rød. Dette ble gjort for å tiltrekke ekstra oppmerksomhet, slik at man ikke skulle være uheldige å slette noe man ønsker å beholde.

Vi har to forskjellige farger på markørene på kartet. Grønn for å vise friske sauer som ikke er fare eller død, og rød for å vise posisjonen til sauer som har avgitt en alarm. I tillegg til de to markørene har vi også en gravstein som popper opp på karet hvis sauen dør.

8. VEDLEGG

8.1 DOKUMENTER

01. Teknisk Brukermanual
02. Brukermanual(ikke-teknisk)
03. Sprinter
04. Møtereferat og Kundemøte
05. Verktøy
06. Product Backlog & Burndown Graf
07. Kravspesifikasjoner
08. Skisser og Diagrammer
09. Risikotabell
10. Burndown Chart

8.2 PROGRAMVEDLEGG

- Kode.zip