# Kap 1 Introduksjon

## Kap 1.1 Om faget

Faget *IT1901 - Informatikk prosjektarbeid 1* handler om gruppearbeid innen IT og utforming av prosjekter i arbeidslivet. En viktig del av dette er forholdet til kunden.  
  
Hovedmålene i dette faget er å gi studentene en forståelse for samspillet mellom produkt og prosses-orientert utfordringer og aktiviteter i programmeringsprosjekter. En viktig del av dette faget er å tilegne seg praktisk og teoretisk kunnskap om hvordan prosjektarbeid innen IT burde foregå, med henhold på bl.a. arbeidsmetoder.

## Kap 1.2 Om oppgaven

Vi “later som” at vi jobber som programmerere i et programvareselskap der vi blir kontaktet av en sauebonde. Sauebonden har allerede digitale sendere til å feste rundt halsen til sauene og det er allerede fastsatt hvordan disse fungerer (hva slags data vi får fra dem og hvor ofte, osv). Fra disse sendere får vi tilsendt posisjonen til sauene og info om sauens hjertefrekvens tre ganger i døgnet og hver gang et angrep registreres. Bonden ønsker vår hjelp til å utvikle et system som ved hjelp av disse senderne hjelper bønder med å registrere, administrere og holde oversikt over sauene sine, bl.a. med henhold til rovdyrangrep. Dette systemet skal være brukervennlig for bøndene det er laget for, som ikke nødvendig har noen spesiell kompetanse innen IT eller PC-bruk. Vi får i oppgave å komme med forslag til et slikt system, basert på analysen vi har gjort i samarbeid med kunden, for å så utvikle dette systemet.  
  
Prosjektoppgaven vår er basert på et ekte prosjekt, men i vårt prosjekt skal vi lage en simulator som simulerer beskjedene vi får fra sendere som våre fiktive sauer har rundt halsen. I prosjektet vårt spiller studass på et vis spiller rollen som kunden/bonden vi utvikler systemet til.

## Kap 1.3 Om gruppen

Vår gruppe 10 består av 6 informatikk studenter som alle studerer BIT, Jørgen Rugelsjøen Wikdahl, Anders Sildnes, Per Øyvind Kanestrøm, Svenn Kvelstad, Lars Erik Græsdal-Knutrud, Tor Økland Barstad. Det tok ikke lang tid før vi fikk samlet oss første gang, og det gikk enda raskere å opprette en felles Facebook-gruppe for prosjektet. Alt i alt var vi godt fornøyd med gruppen og kom fort i gang med prosjektet.  
  
Da et av hovedmålene i dette faget er samarbeid i gruppe, så vi det ikke som et problem at enkelte på gruppen har større kompetansenivå enn andre, siden vi som gruppe skal løse oppgaven sammen og ikke individuelt.

## Kap 1.4 Gruppas medlemmer

Tor Økland Barstad  
Alder: 22, Hjemsted: Ås  
Ansvarsområde: Progammering  
  
Tor hadde fra starten av god forståelse for programmering innen Java og objektorientering. Han hadde også litt erfaring med klasse- og ER-diagrammer og har vist god vilje til å tilegne seg kunnskap om andre typer diagrammer som vi har hatt bruk for. Sammen med Jørgen har han stått for det meste av diagramproduksjonen i starten av prosjektet.  
  
Anders Sildnes  
Alder: 19, Hjemsted: Asker  
Ansvarsområde: GUI  
  
Anders har en bred bakrunn innen web- og it-utvikling og har gjennom sine engasjement i ITK Samfundet og IUG NTNU tilegnet seg nyttige programmeringskunnskaper som gruppa har kunnet nyte godt av. Han har en grundig forståelse for hvordan man programmerer effektivt og har erfaring fra Java såvel som AS3 og C++ fra før. Han har også hatt erfaring med bruk av Qt4 så han fikk ansvaret med å utvilke det grafiske brukergrensesnittet. Dette har han gjort på en strålende måte.  
  
Per Øyvind Kanestrøm  
Alder: 21, Hjemsted: Stavanger  
Ansvarsområde: Scrum-master. Kart.  
  
Som ScrumMaster har Per Øyvind fungert veldig bra. Han har vist gode evner til å kommunisere ikke bare med gruppa, men også med «kunde» (stud.ass/fagstab). Han har god evne til å holde oversikten over hva som må gjøres og får folk med seg på det. Som medansvarlig for kartdelen har han vist en god evne innenfor “research” til å finne metoder og klasser som kan brukes. Erfaring fra C#, Silverlight og Windows Forms har også gitt en god bakgrunn for å kunne sette seg inn i kode og har gitt ham en bra forståelse for hvordan et program er bygget opp.  
  
Lars Erik Græsdal-Knutrud  
Alder: 21, Hjemsted: Vestnes  
Ansvarsområde: Database  
  
Lars Erik har vært uvurderlig for hvordan gruppa har klart å produsere kode. Han har vist en stor vilje til å jobbe med koden og har bred erfaring fra systemutvikling, SQL-databaser og generell programmering. Han har vært et arbeidsjern som har stått på og programmert mye av både databasedelen og logikken som ligger til grunn for programmet. Han hadde også bra forståelse for ER- og UML-modellering.  
  
Jørgen Rugelsjøen Wikdahl  
Alder:21, Hjemsted:Trondheim  
Ansvarsområde: Dokumentasjon  
  
Jørgen har bidratt med god vilje til å jobbe og lære, og har sammen med Tor hatt ansvar for forskjellige diagrammer. Han har også hatt overoppsynet med dokumentasjon og raportskriving, og har vist evne til å få folk til å gjøre det som må gjøres for at prosjektet skal komme i havn. Grunnleggende erfaring i Java og objektorientering, samt litt erfaring med databaser har kommet godt med utover i prosjektarbeidet.  
  
Svenn Kvelstad  
Alder: 33, Hjemsted: Trondheim  
Ansvarsområde: Kart  
  
Svenn hadde erfaring med prosjektarbeid og rapportskriving før i tillegg til grunnleggende kunnskaper innen Java og objektorienterte metoder. Svenn tok på seg oppgaven med å integrere kart inn i systemet, og var nødt til å sette seg inn i hvordan dette fungerte, siden ingen hadde erfaring med å integrere kart inn i IT-systemer fra før. Han har også driftet nettsider og jobbet med grunnlegende databaser som kom godt med i prosjektet.

## Kap 1.5 Valg av prosessmetode

Vi valgte å bruke Scrum-metoden, slik vi ble anbefalt fra faglærer. Det ble det sagt at studenter som hadde kjennskap til andre prosessmetoder de heller ville bruke hadde muligheten til det, men siden ingen av oss hadde noe særlig erfaring med prosessmetoder, var det ikke et alternativ å velge en annen prosessmetode enn Scrum.  
  
Vi fikk presentert Scrum av faglæreren i en av de første forelesingene som ble holdt, men søkte også etter informasjon på egenhånd. Ingen av gruppemedlemmene hadde erfaring med Scrum og vi brukte noe tid i starten på å bli kjent med denne arbeidsmetoden og venne oss til å gjøre ting på Scrum-vis.  
  
Vi startet først med å bruke scrumtavle i Trello (se kap. 10), men byttet fort over til IceScrum (se kap. 10). Der hadde vi en oversiktlig tavle hvor vi fikk oversikt over alle oppgaver som lå til behandling og var under arbeid, og hva som måtte prioriteres til neste gang. Denne metoden brukes også av mange IT-selskaper verden rundt når de jobber tett med kunder, siden kunden da lett kan følge med på prossesen gjennom scrumtavlen, komme med innspill og være med gjennom hele prosjektet.

# Kap 2. Tidsestimering

I dette prosjektet har gruppen måttet lære seg mange nye grensesnitt og modeller å jobbe etter. Ingen på gruppen hadde tidligere hatt prosjektarbeid i IT. Tidligere studenter hadde ikke hatt samme type oppgave som det vi fikk, så de kunne ikke komme med råd for å hjelpe oss. Tidsestimeringer ble med andre ord vanskelig å gjøre nøyaktig. Til og med kunden vår måtte innrømme å ha bommet litt i tidsestimatet for prosjektet og kravspesifikasjonen ble krympet etter en måned.  
  
Vi startet prosjektet vårt med å lage en «Work Breakdown Structure»[[1]](#footnote-1), WBS, til prosjektet. Denne laget vi i form av en liste over ting som måtte bli gjort innen prosjektet hadde deadline. Dette ga oss en oversikt som skulle hjelpe oss å få en anelse over hvor langt vi var i prosjektet til enhver tid. På denne måten kunne vi også bli hjulpet til å lage realistiske tidsestimeringer senere i prosjektet. Det kan for eksempel være ønskelig å bruke mye tid på å lage ekstra funksjonalitet til databasene, men dersom man ser at vi i forhold til WBS-listen vår ligger litt etter i prosjektet, kan det hende man innser at annen funksjonalitet må prioriteres først.  
  
I de første sprintene valgte vi å gjøre tidsestimater felles, slik SCRUM anbefaler. Vi gjorde opp estimater i antall timer, men valgte ikke å dele inn i vanskelighetsgrad. Vi følte ikke at dette var nødvendig da vi ikke var godt nok kjent med det vi jobbet med fra før. Estimatene ble gjort felles ved at vi leste opp en oppgave og lot alle på gruppen skrive ned antall timer de antok trengtes. Etterpå leste alle på gruppen opp sitt tall. Den personen som førte inn sprinten bestemte seg så for hva gjennomsnittet lå på, og skrev dette ned. Denne varianten var ment for å simulere SCRUM sin planning poker, utenom at vi ikke lagde “pokerkort” å spille med.  
  
Tidsestimatene viste seg etterhvert å være unøyaktige, og de ble i økende grad vanskeligere å gjøre ettersom kompleksiteten til oppgaven var ukjent for resten av gruppen. Derfor ble det til at vi gikk over til å selv gjøre tidsestimat innenfor de oppgavene vi gjorde selv. Her ble det slik at når man tok på seg en sprint-oppgave skrev man selv ned antall timer man beregnet direkte inn i sprinten. Et unntak er rapportskriving, hvor alle hadde forutsetning for å gjøre tidsanslag. Da gjentok vi praksisten med planning poker ved skrive ned tall, for så å lese opp.  
  
Et stykke ut i prosjektet ble det klart for oss at vi hadde mye resterende arbeid, og det ble i tillegg lagt vekt på at ikke alle medlemmene i gruppen gjorde like mye arbeid i hver sprint. Vi bestemte oss derfor for en regel om at vi skulle lage hver sprint til å ha 120 arbeidstimer, eller 8 timer per person hver uke.  
  
Idéen virket god, men det ble også til at flere på gruppen ikke klarte å holde seg til tidsrammene sine. Sprintene måtte heretter forskyves med flere dager. Vi reagerte med å sette ned kravet igjen slik det var originalt. På denne måten ble burnup-chartene våre mer nøyaktige og penere å se på, og vi ble ferdig med sprintene våre til planlagte tidspunkt.  
Det har vist seg at tidsestimering er vanskelig å få til korrekt når man ikke har erfaring med slik jobbing fra før, men vi føler vi har lært mye om hvordan dette bør gjøres gjennom prosjektet, og det gjenspeiles i hvordan sprintene ble gjennomført i slutten av prosjektet kontra starten.

# Kap 3. Prosjektorganisering

## Kap 3.1 Ansvarsområder

Til å begynne med identifiserte vi hva de forskjellige ansvarsområdene ville være. Ansvarsområdene vi kom frem til var GUI, databaser, dokumentasjon, kart og prosjektets fremdrift (Scrum-master) (se kap 1.4). Vi gikk på rundgang med å si tankene våre om å ha de forskjellige ansvarsområdene vi hadde identifisert, både med tanke på erfaringen og kompetansen vi hadde og også hvordan vi trodde vi ville likt å ha et slikt ansvarsområde. Basert på dette fordelte vi ansvarsområdene.  
  
Det ble til at de som hadde kompetanse innen spesielle områder (se kap. 1.4) fikk ansvaret for disse. Dette var noe alle syntes virket fornuftig, inkludert gruppemedlemmene som tok på seg de gjeldene ansvarsområdene.  
  
Flere på gruppen sa at de kunne ta rollen som Scrum-master, men at de foretrakk å ha andre ansvarsområder i stedet. Per Øyvind tok ansvaret og rollen som Scrum-master og det skulle vise seg at det var et godt valg.  
  
Selv om Per Øyvind var Scrum-master tok han også med-ansvar innen andre områder som kart og GUI. Per Øyvind fylte ansvaret som Scrum-master slik han skulle, men det betyr ikke at han er den eneste som har tatt initiativer.

At man har ansvar for noe betyr ikke at man må gjøre det alene, men det betyr at man har ansvar for å passe på at det blir gjort.

# Kap 4 Arkitekturforklaring

under systemdesign

Database

Klient

Server

Front end

Programmet er skrevet i programmeringspråket Java. Vi har forsøkt å implementere det i MVC stil.   
  
Egentlig skulle dette prosjektet bli helt plattformuavhengig, men på grunn av et teknisk valg av bruk av en Java Swing komponent for kartvisning i programmet så er programmet avhengig av Windows plattformen.  
Qt Jambi krever også at Java versjonen som blir brukt er 32bit.  
  
Den grafiske delen er håndtert Qt Jambi.  
  
Kravspesifikasjonen er altså;  
- Java 32bit  
- Windows 32/64 bit  
  
Kun testet på Windows 7 32bit/64 bit og Windows 8 64bit.

Back end

Databaser + server

# Kap 5. Risiko i prosjektet

For en generell risikoanalyse se vedlegg 1 (Risikoanalyse).

## Kap 5.1 Kommunikasjon og konflikt

Noe som kan slå feil når man jobber på gruppe er at man på grunn av mangel på god kommunikasjon gjør oppgaver på nytt eller på feil måte. En ting vi har gjort for å unngå dette er såklart å ha møter, men vi har også arbeidet mye sammen, slik at det bare har vært å spørre hvis man har lurt på noe. Vi har også gjort god bruk av klassediagrammer for å sørge for at vi har en lik forståelse av hva vi skal lage og hvordan det skal henge sammen, og vi har alltid hatt muligheten til å ta kontakt via Facebook eller mobil hvis det er noe som må tas opp.  
  
Kanskje nettopp fordi vi har hatt god kommunikasjon har vi har verken sett noen antydning til splitting av gruppen eller tiløp til konflikter. Hvis gruppen vår hadde opplevd problemer som splitting så hadde dette mest trolig vært et tegn på samarbeidsproblemer og kommunikasjonsfeil. Dette har vi unngått ved å håndtere uenigheter og diskusjoner på en god måte og generelt kommunisert godt. Hvis vi hadde hatt konflikter eller tiløp til konflikter innad i gruppen ville vi først prøvd å løse disse ved å snakke sammen om dem på en konstruktiv måte, men som nevnt har ikke dette vært en aktuell problemstilling for oss.

## Kap 5.2 Feilestimering og deadlines

Som faglærer advarte om forekom det, spesielt under de første sprintene, at vi overvurderte hvor mye vi kunne få gjort og undervurderte hvor lang tid ting ville ta. Det er flere ting som kan føre til at ting tar lengre tid å få til en forutsett:

*Feilberegning.* Man har generelt en tendens til å undervurde hvor lang tid man bruker på å utføre en oppgave.[[2]](#footnote-2)

*Andre ting.* Vi er studenter som har andre fag hvor deadlines er innvolvert og flere av oss har også ting på siden av studiene, og her kan det også hope seg opp arbeid. Riktig nok har vi prioritert dette prosjektet høyt, og tatt frister alvorlig selv når de har blitt satt av oss selv.

*Utfordringer.* Det hender man må gjennomføre en oppgave som man ikke har forutsett eller tatt med i beregningene, eller at en oppgave viser seg å være mye mer utfordrende enn antatt. Denne risikoen var ekstra stor for oss siden vi hadde så liten erfaring fra før og gjorde så mange av oppgavene for første gang.

For å redusere risiko i fht feil tidsestimering kunne vi i større grad ha forhørt oss mer med individer som har gjort lignende prosjekter, men det ville tatt tid om vi skulle gjort det på en aktiv og gjennomgående måte. En ting vi gjorde var å runde opp tidsestimatene når vi var i tvil, slik vi ble anbefalt. Likevel skjedde det fra tid til annen at vi ikke rakk deler av oppgavene innen deadlinsene.

En annen ting vi bevisst gjorde med tanke på risikoreduksjon var å prøve og bli ferdig med ting så fort som mulig. Vi satte også som mål å være ferdig med største delen av prosjektet to uker før innlevering. Da er man mindre sårbar for uforutsette utfordringer, og selv hvis man i liten grad støter på uforutsette utfordringer får man mer tid til finpussing på- og bug-testing av programmet og til rapportskriving. Det viste seg at det var lurt å ha denne bufferen mtp hvor mye tid som gikk bort på kartdelen av prosjektet. (se kap 5.7)

## Kap 5.3 Misting av arbeid

Vi har brukt Git (se kap. 10) til å koordinere og lagre kildekode. Dette gjør at alt vi gjør ligger lagret på et eksternt og trygt sted. Git gir oss også mulighet til å hente ut en eldre versjon av programmet dersom noe skulle gå galt med den versjonen vi jobber på. Vi hadde et tilfelle der vi var nødt til å gå tilbake til en ukegammel versjon, noe som gjorde av vi mistet mye arbeid og kode.

Mens vi fortsatt ikke hadde en fungerende versjon av programmet jobbet vi kun i en “branch”[[3]](#footnote-3) i Git. Det forekom i denne lange perioden at vi måtte gå tilbake til tidligere versjoner, men vi klarte oss bra med alt det Git automatisk gjør for å lagre tidligere versjoner av arbeidet vårt. Så snart vi hadde en fungerende versjon av hele systemet ble denne versjonen lagret som en “branch”, mens noen av oss jobbet med forandringer og oppdateringer på andre brancher. Slik unngikk vi å ødelegge den fungerende versjonen. Så snart vi begynte med “branching” sørget Lars Erik for å lære seg det og deretter lære det bort til resten av oss.

## Kap 5.4 Sykdom og medlemmer som ikke møter

Tidlig ut i prosjektet vårt skrev Svenn: “Er det noen som kan bli med meg å finne ut hvordan vi kan integrere Kartverkets karttjeneste i programmet vårt? Jeg kjenner jeg trenger en til til å være med og grave, og så er det sikrere å ha to på det i tillfelle jeg blir syk.” Senere skjedde nettopp dette: Svenn ble syk i en kort periode. Studass anbefalte oss også å være minst to på hver oppgave, og vi fikk til dette overalt med unntak av databaser og server-oppsett.  
  
Medlemmer som ikke møter har ikke vært noe problem for oss. Alle har satt av tid til å møte opp på møter og møtt opp hvis ikke annet har blitt meldt ifra. I de tilefellene der personer på gruppen ikke har møtt opp, har den aktuelle personen hatt en god grunn og sagt fra i god tid. Det har heller aldri vært noe problem å få kontakt med noen i gruppa utenom møter når vi har ønsket det.

## Kap 5.5 Forsentkomming

Først flere uker ut i prosjektet identifiserte vi en tendens til at det generelt sett var flere medlemmer som kom for sent til møter. Dette ble tatt opp i et SCRUM-møte, der vi fastslo at tendensen ikke hadde hatt noen skade for prosjektet. Vi diskuterte immidlertid hvordan dette kunne påvirke prosjektet dersom tendensen enten fortsatte eller vokste mer. Konklusjonen ble at vi ga en felles verbal notis til hverandre der vi sa oss enige om å forsøke enda hardere på å møte opp til rett tid. Vi følte det ikke var nødvendig med noen strengere måte å behandle dette på, ettersom tendensen fortsatt ikke hadde gjort noe skade på prosjektet. Vi ble også felles enige om at ingen med intensjon hadde møtt opp forsinket, men heller vært uheldig med å kikke på klokken og lokalisering av møterom (se kap. 11.1). Det ble normert en felles aksept for at uhell kan skje, så streng etterfølging ble heller ikke gjort dersom noen fortsatt møtte opp til feil tidspunkt, så lenge det ble begrunnet og beklaget for.

## Kap 5.6 Avveining mellom forhåndsregler og effektivitet

Vi har tatt forhånsregler for å redusere risiko, men ikke så mange forhåndsregler som vi kunne ha gjort. F.eks har det blitt til at man i stor grad jobber på hver sin del til tider, siden det er mer effektivt. Dette gjør at vi er mer sårbare hvis noe skulle skje som hindrer et av gruppemedlemmene å bidra videre på en ordentelig måte, men samtidig bør man ikke optimalisere prosjektet med henhold til *kun* risiko-reduksjon for enhver pris. Sagt på en annen måte: Hvis tiltak mot risiko blir så drastiske at de fører til betydelig treigere fremgang gjør dette at sjansen for at vi ikke blir ferdig i tide større, som også er en risiko. Derfor kan man noen ganger være nødt til å gjøre avveininger. Samtidig er det mange tiltak mot risiko som ikke krever arbeid eller gjør ting mer slitsomt, men rett og slett dreier seg om å gjøre ting på en smartere måte.

## Kap 5.7 Manglende kunnskap på spesielle områder.

Prosjektet har vært stort og omfattende og vi har måttet sette oss inn i mange metoder, verktøy og teknikker vi ikke hadde jobbet med før. Dette har tatt mye tid.  
  
Gjennom største delen av prosjektet jobbet vi opp mot JXMapViewer[[4]](#footnote-4). Det skulle dessverre vise seg at det var veldig mye jobb med å få dette til å fungere skikkelig og vi støtte på flere problemer, bl.a. i forbindelse med at kartet ikke beholdt størrelsen etter at størrelsen på vinduet det lå i ble forandret på. Derfor følte vi at det var mer produktivt å gå over til Google sin karttjeneste og API (se kap. 10) for så å servere kartet med sauedata og posisjoner til programmet vårt gjennom et javascript. Byttet av karttjeneste skjedde en uke før prosjektet skulle være ferdig.Det er langt fra optimalt å bytte karttjeneste så sent i prosjektet, men vi var av den oppfatningen av at vi ville kunne levere et bedre produkt vha. Google Maps enn om vi hadde fortsatt med JXMapViewer.

# Kap 6 Produktbeskrivelsen

## Bruksområder

Steg Beskrivelse 1

1. Bruker logger inn på systemet, ved hjelp av riktig brukernavn og passord. Hvis dette er riktig blir man overført til steg 3. Er det ikke riktig, steg 2.
2. Kontakt administrator for brukernavn og passord.
3. Bruker er logget inn på system, her kan han browse kar, sauer, og se alarmer.

Steg Beskrivelise 2

1. Add sheep funksjon, her går bruker inn på eget GUI for å legge til ny sau(er). Legger til navn, dato født, vekt, farm og om den er i live.
2. Det blir lagt til en ny sau, ny ID genereres automatisk for at 2 sauer har samme ID.
3. (blir sau markert i sheep list? Blir så kanskje markert på kart?)

Steg Beskrivelise 3

1. Message funksjon her får bruker info om sau (farm, dato født), siste alarm, location.

Steg Beskrivelse 4

1. Alarmfunksjon. Her får bruker inn alle nylige alarmer.
2. For å behandle en alarm, må bruker ha sett på og godkjent alarmen.
3. Alarm forsvinner fra alarmlisten og bruker returneres til hovedvindu

Steg Beskrivelse 5

1. Redigere brukerfunksjon. Her kan du endre brukerens rettigheter.
2. Skifte farm-rettigheter, endre personalinfo.
3. Endringene godkjennes og bruker returneres til eget vindu

Steg Beskrivelse 6

1. Sau blir angrepet, hjerterytme forandres.
2. Tekstmelding blir sendt.
3. “Beskrivelse 1”
4. “Beskrivelse 4”
5. Bruker sjekker sau, endrer info hvis den er død.
6. Er sau død, blir satt som død i databasen og på kar. Hvis ikke se bort fra 6.

**(er ikke i nærheten av ferdig her)**

# Kap 7 Prosessbeskrivelse

## Kap 7.1 Sprint

Til å begynne med måtte vi bestemme varighet på sprintene våre. Vi definerer en lang sprint som ca 3-4 uker og en kort sprint som ca 1-2 uker.

|  |  |
| --- | --- |
| **Fordeler med lang sprint** | **Fordeler med kort sprint** |
| Mer tid brukes på programmering framfor møter[[5]](#footnote-5) | Kontinuerlig oversikt over prosjektet. |
| Med mer tid kan man potensielt legge mer fokus på kvalitetsløsninger framfor å kun fokusere på at kode skal fungere. | Lettere å unngå Parkinsons Law[[6]](#footnote-6). (*“Work expands so as to fill the time available for its completion.”*) |
|  | Dersom noen blir syke vil det være enklere å plukke opp arbeidsoppgavene til denne personen. |
|  | Lettere å unngå utsetting (procrastination). |
|  | Ettersom man jobber iterativt blir det enklere å oppdage forbedringer til programvare underveis. |

Nøkkelordene til korte sprinter er oversikt og mer fleksibilitet. Gruppen hadde dog kun mulighet til å samles i fellesskap *fast* én gang i uken. Hvis dette møtet skulle brukes til sprint-planlegging ville vi ha liten tid samlet til å diskutere kode og få hjelp av hverandre. Dette gjorde at gruppen bestemte seg for sprinter på 2 uker i lengde. Oversikten var viktig å få med tanke på at gruppens mangel på erfaring.

## Kap 7.2 Arbeidsfordeling

I kapittel 3.1 skrev vi om annsvarsområder. Annsvarsfordeling er ikke det samme som arbeidsfordeling, men likevel ble det til at mange arbeidet mest med sitt annsvarsområde. Et klart unntak fra dette er Per Øyvind som var Scrum-master men som likevel gjorde en betydelig del av arbeidet utover organiseringen av gruppa.  
  
Når et ansvarsområde har vært for mye for en person er det flere som har jobbet sammen på oppgaven. Dette har også vært gjort med tanke på risikoreduksjon. For eksempel er det flere som har jobbet på diagrammer, flere som har jobbet på kart og flere som har jobbet sammen på GUI. Alle har vært med på å skrive dette dokumentet og på forbredingen av midtveispresentasjonen.

## Kap 7.3 Møter

Vi ble enige om at hver uke skal inneholde en til to møter. Årsaken til at vi kom frem til dette er at vi ikke har tid til å sette av flere sprintmøter i uken. Dette kom veldig synlig frem når vi brukte nettjenesten doodle.com og så at det ikke mange økter som ikke overlappet. Onsdager klokken tolv ble valgt som fast møtetid.  
  
I tillegg til dette har ekstraordinære sprintmøter samt arbeidsøkter blitt kjørt når det har passet for de fleste. Det har vært vanligere at en uke har inneholdt et slikt ekstraordinært sprintmøte enn at dette ikke har vært tilfelle. Tirsdag har vært en populær dag for slike møter. Det har også forekommet at to gruppemedlemmer som har jobbet på en felles del av prosjektet har møttes for å arbeide sammen.

## Kap 7.4 Kildekodedeling

Som tidligere nevnt brukte vi Git til både kildekode og andre filer (diagrammer, dokumenter, osv). Dette er et program som bare noen på gruppa kjente til fra før og som tok litt tid å lære, og de som kunne Git fra før hjalp til med å lære opp resten. Etter ikke så lang tid var alle komfortable med Git og arbeidet gikk mye lettere enn det hadde gjort uten et slikt hjelpemiddel. I det hele tatt kan Git være et veldig praktisk og kraftig verktøy for å koordinere fells IT-prosjekter.

Kodingen gjorde vi i stor grad i Eclipse (kap 10.5). Eclipse er et verktøy som egner seg godt for dette, og alle var kjent med Eclipse fra tidligere kurs. Vi hadde en felles prosjektmappe i Eclipse sin “Project explorer”-sidebar som ble oppdatert via Git på samme måte som de andre mappene i vårt felles Git-repository. Så vi fikk delt kildekoden på en automatisert og effektiv måte.

## Kap 7.5 Facebook

Innad i gruppa har vi valgt Facebook (kap 10.6) som kommunikasjonsmiddel for spørsmål, møtekalling og andre relevante og irrelevante kommentarer og opplysninger. Vi opprettet en privat Facebook-gruppe slik at det kun var vi som kunne se den informasjon vi delte.  
  
Hovedgrunnen til at vi valgte å bruke Facebook er at alle på gruppen oftest har denne nettsiden lett tilgjengelig eller får notifikasjoner på mobilen når det skjer noe nytt der.  
En annen fordel er at man kan se hvem som har sett hva. Da vet man fort om alle har fått med seg den aktuelle informasjonen, eller om man må bruke andre kommunikasjonsmidler for å få tak i personen som ikke har fått det med seg. Dessuten er det veldig lett å nå ut til hele gruppa.   
Et problem kan være med gruppesystemet til facebook kan være at det blir uoversiktelig og rotete med mange brukere og man får en høy terskel for å poste noe, men i en gruppe på seks stykker har ikke dette vært noe problem.

# Kap 8 Systemdesign

Her må vi henvise til UML digrammer og slikt  
arkitektur, komponenter, moduler, grensesnitt, og data for et system for å tilfredsstille spesifiserte krav.

## 8.1 Logical design

Ref bilde Lars erik tegna på tavla

## Fysisk design

1. User Interface Design

I dette systemet gir brukeren noe information til systemet så systemet kan generere en output tilbake til brukeren. For eksempel log in screen, her skriver bruker inn brukernavn og passord ved hjelp av tastatur. Det blir sjekket opp imot databasen for riktig passord og brukernavn, så blir bruker sendt videre inn i systemet hvis dette var riktig. Henholdsvis inne på systemet bruker brukeren mus og klikker seg rundt på forskjellige widgets(se kap 10.7).

1. Data Design

Data i sau systemet vårt er representert i graffiks grensesnitt for brukeren, dette har kontakt med serveren som henter ut data fra databasen. I databasen så lagres all data som er hensiktsmessig for systemet vårt(sauer, farmer mm.).

1. Process Design

For at det skal sendes en request til serveren hender det at inputen må valideres lokalt før den sendes videre. For eksempel passer man på at man skriver inn brukernavn og passord i det hele tatt før man sjekker med serveren om brukernavn og passord er riktig, og man passer på at nye sauer ikke er mindre enn null år gamle.

# Kap 9 Implementasjon

I dette kapittelet vil vi liste opp alle klassene som systemet vårt består av, og hvilken rolle de spiller i systemet. Det vil bli en ganske overfladisk beskrivelse av hva klassene faktisk gjør, så for mer info vil vi referere til UML-diagrammet i vedleggene.

## Kap 9.1 Klient-systemet

## Kap 9.2 Server-systemet

##### com.db:DatabaseConnector

Interface til databasen. Inneholder public metoder som oversetter til SQL-spørringer, og private hjelpemetoder som forenkler det å legge til nye metoder i klassen

##### com.net:ClientHandler

ClientHandler lager en ny unike instans hver gang en bruker logger på. Den tar imot forespørsler og sender svaret tilbake til klienten. Den sender forespørsler til databasen og deretter svar tilbake enten med data eller beskjed om operasjonen var vellykket.

##### com.net:MailNotifier

Logikk for å koble seg til google sine smtp-servere og sende epost. Blir kallt av AlertNotifier hver gang en alarm blir mottatt av simulatoren.  
  
com.net:Request  
Wrapper-klasse for forespørsler til serveren fra klienten.Kan settes også i et av to modus; forespørsel eller logg av. Logg av gjør at serveren kan lukke tilkoblingen uten feilmeldinger.  
  
com.net:Response  
Wrapper-klasse som inneholder svaret som blir sendt tilbake til klienten fra serveren.

com.net:Server  
Server-klassen er basisen for hele serversiden av programmet. Den starter opp AlertNotifier, SheepStatusSimulator og fungerer som base for ServerGUI. Det er denne klassen som mottar tilkoblinger fra klienter via sockets, og starter ClientHandler for hver av disse.   
  
com.net:ServerGUI  
ServerGUI er et GUI som bygger på Server-klassen for å gjøre det lettere å administrere databasen etc.  
  
com.skype:PhoneNotifier  
G gjør det mulig å varsle brukere på sms når systemet mottar en SheepAlert. Den er bygget på en tilpasset versjon av skype sin SDK. Starter opp runtimen til skype, og sender sms til brukere når den får beskjed om det.

##### core:AlertNotifier

Sørger for at brukere får beskjed som en alarm uavhengig om de er logget på klient-systemet.   
  
core:HelpPrinter  
Statisk hjelpeklasse som inneholder hjelpeteksten til kommandoene i ServerGUI for å holde ServerGUI så ryddig som mulig.  
  
core.classes.\*  
Se implementasjon for klient-systemet.  
  
core.settings.\*  
Se implementasjon for klient-systemet.  
  
core.sim:SheepStatusSimulator  
Simulerer statusene fra sauene. Startes av Server-klassen, og lager nye statuser for alle levende sauer på et gitt, redigerbart, intervall. I tillegg tar den seg av generering av alarmer og sender disse til AlertNotifier.  
  
core.sim:DatabasePopulator  
Et mellomledd mellom ServerGUI og DatabaseConnector. Den er i hovedsak laget for å generere store kvanta av data og deretter sette de inn i databasen via DatabseConnector. Den styres via kommandolinjen til ServerGUI.

**Kap 10 Verktøy**

#### Javadoc

Dokumentasjonsgenerator for generasjon av API-dokumentasjon i HTML-format fra kildekode i Java.

**ER-Modeller**

Program for modellering av databaser og ER-diagrammer

**Dia**

Program for design av diagrammer med støtte for bl.a UML-diagrammer.

**UML**

Unified Modeling Language (UML) ble brukt i produksjonen av diverse diagrammer.

**Eclipse**

Et “Integrated Development Environment” laget for Java-utvikling.

**Facebook**

Sosial mediaplatform med innebygget funksjon for gruppekommunikasjon og informasjonsdeling. Har vært svært viktig for hvordan gruppen har samarbeidet.

**GIT**

Distribuert versjonskontrol- og kildekodesystem.

**Qt Jambi**

Qt Jambi er et Java bibliotek for å lage grafiske brukergrensesnitt med bindinger mot C++ biblioteket Qt. Vi valgte Qt som verktøy for GUI-design siden Anders allerede hadde erfaringer med det fra før.

**Kartverkets WMS-tjeneste**

En visningstjeneste for kartbilder over http-protokollen.

**Google maps API**

Interface for å integrere Google sin kartløsning på nettsider.

**Trello**

Trello.com er en nettside som hjelper folk med å få oversikt over og koordinere prosjektene sine

**IceScrum**

Et online, serverbasert system for organisering og gjennomføring av Scrum-baserte prosjekter.

**MySql**

Et open source, relasjonsbasert databasesystem.

**Google docs**

Rapportskrivingen og i forbredelsen av midtveispresentasjonen, samt i planleggingen av enkelte av sprintene, har vi brukt Google Docs. Dette har veldig praktisk siden dokumentene lagres automatisk, og man kan skrive og redigere på det samme dokumentet samtidig, og se hva de andre gjør i real-time.

# Kap 11 Evaluering

## Kap 11.1 Hva gikk bra

*Bra sammerbeid*

Vi har hatt et utmerket samarbeid i gruppen. Som nevnt har det ikke vært noen konflikter eller missnøye. Vi har jobbet jevnt gjennom hele prosjektet og hatt et godt fungerende system med ansvarsområder. Hvis noen har trengt hjelp på enkelte områder har de ikke nølet med å spørre om hjelp. Vi har hatt gode møter og har som nevnt kommunisert godt og effektivt gjennom Facebook (kap 10.6).

*Alle var arbeidsvillige*

Det stod ikke på arbeidsviljen til folk under dette prosjektet. Alle har vist engasement og innitiativ og vært villige til å lære og jobbe. Vi har måttet lære nye ting både for å gjennomføre oppgavene våre (Qt Jambi, Dia, osv) og for å koordinere arbeidet (Git, IceScrum, osv), men alle har ønsket å lære det de trengte. Enkelte på gruppen har kunnet mer på noen områder enn andre, og disse har vært flinke til å lære bort til andre.

*Strukturerte i startfasen og kom fort igang*

Vi fikk en god start, både sosialt og prosjekt-messig. Alle virket engasjerte og innsatsvillige. Vi lagde Facebook-gruppe med en gang, delte ut ansvarsområder hurtig, og kom fort igang med planleggingen av systemet.

*Flinke til å bruke Scrum*

Fra første møtet av var vi klare på at en viktig del av dette faget var prosessmetoden, og at vi være bevisste på arbeidsmetodikk. Vi var kjappe med å sette oss inn i Scrum og komme til enighet om hvordan vi kunne implementere Scrum i prosjektet vårt.

*Romreservasjon*

Romservasjon (kap 10.17) hjalp oss veldig. Vi fikk fred og ro i møtene våre, noe som hjalp oss med å kommunisere og arbeide mer effektivt. Et lite problem som vi kanskje kunne gardert oss mot noen ganger var at noen av rommene var vanskelige å finne, noe som gjorde at noen av oss kom for sent til enkelte møter. Vi var flinke til å bestille rom vi hadde vært på før når de var ledige, men vi kunne kanskje ha bestilt rom tidligere, i og med at vi hadde faste Scrum-møter.

IceScrum

Selv om IceScrum ikke er et perfekt program er vi godt fornøyde. Det var godt å ha et program som gjorde så mye av tilpasningen til Scrum for oss. Lars fikk satt det opp på sin egen server, og det fungerte smertefritt.

## Kap 11.2 Hva gikk dårlig

*Alle endte ikke opp med å gjøre like mye*

Noen hadde mer kompetanse på sitt område enn andre, både på grunn av kompetansen de opparbeidet seg underveis i prosjektet og kompetansen de hadde fra før. Dette gjorde at de var i stand til å gjøre mange ganger så mye per tid på dette området enn andre. På flere områder ville det for mange av gruppens medlemmer tatt veldig mye arbeid bare å komme til det punktet hvor man kan lage noe i det hele tatt. Selv om det forekom at folk underveis i prosjektet satte seg inn i helt nye områder slik at flere kunne jobbe på dette området var det ikke bare å flytte vilt rundt på hvem som gjorde hva slik at arbeidet ble jevnt fordelt. For eksempel ble det til at Lars Erik gjorde mer arbeid enn snittet bl.a. siden han var den eneste som fra før hadde hatt kurs i databaser.  
Vi prøvde etter beste evne å fordele oppgaver og ansvarsområder slik at det ble jevnt, og gitt utfordringene med å fordele arbeidet jevnt synes vi ikke vi gjorde dette så aller værst. Noen har programmert mer enn andre, mens andre har laget flere diagrammer enn andre og jobbet mer med rapporten enn andre. Ingen har endt opp med å bruke mindre tid på prosjektet enn man typisk bruker på et “Gløshaugen-fag”. Vi er takknemmelige ovenfor de som til sammen har gjort mer enn snittet av gruppa, og kan ikke se noen måte det ville vært praktisk mulig med våre forutsetninger å gjøre det slik at alle gjorde like mye.

*Litt dårlig start med dokumentasjon av arbeid*

Det var flere på gruppen som hadde erfaring med Trello fra tidligere arbeid, så vi startet med Trello. Trello var utmerket for vårt prosjekt og vår arbeidsmetodikk - tenkte vi. Det ble imidlertid en del arbeid med å tilpasse Trello til Scrum, eksempelvis måtte vi generere våre egne burndowncharts (og div annet). Når vi fikk byttet til IceScrum og lært oss å bruke det ble ting bedre.

*Burndown-chartene kunne vært penere*

Dette har nok å gjøre med at ikke alle alltid har vært like flinke til å jobbe jevnt, men det er ikke bare det. I begynnelsen var ikke alle like flinke til å skrive ned timer når de skulle gjøre det, og derfor ble det til at man tok av mange av timene på en gang mot slutten av sprinten, uten at det betyr at alt arbeidet ble gjort da. En annen grunn er at at vi på noen av sprintene begynte å jobbe rett etter at sprinten hadde startet, og tok av timer samme dagen. Når vi gjorde dette ble det ikke registrert som arbeid vi hadde gjort, men heller som om vi hadde startet med en mindre oppgave forann oss, men dette ble vi først bevisst på senere. Dessuten er at helgene er med i sprintene, som fører til at grafen blir flatere til tider. En annen ting som har gjort burndown-chartene mindre penere enn vi hadde ønsket er at vi var ganske nybegynnere på å estimere timer, spesielt i begynnelsen. Vi estimerte for lite timer på enkelte oppgaver, og ble ikke ferdig med disse innen timesanslaget. Det hendte også at vi brukte betydelige mengder med tid på ting som ikke hadde blitt spesifisert som oppgaver i sprinten.

*Ble ikke ferdig så tidlig som vi hadde tenkt*

Vi ble ikke helt ferdig med programmet så tidlig som vi ønsket, her støttet vi på noen uforutsette problemer som tok lenger tid å løse enn hva vi hadde forventet oss. Dette gikk jo utover debuggings prossesen og testingen av systemet, som gjorde at vi fikk mindre tid til dette.

*Møtte på problemer (men det var vel mest uflaks?)*

qt var vanskalig (mer research?)  
Qt var vanskelig, men det viste seg å ha innebygget qt design program som generer kode for deg når du gjør designet ditt slik du vil at det skal være. Gjør det mye enklere enn å kode fram designet du vil ha. for å unngå problemer, kunne vi brukt det anbefalte programmet netbeans. Men det var utrolig artig å lære seg et annet rammeverk, enn det som ble anbefalt. Da måtte vi gjøre reasearch selv, og lære alt på egenhånd.

## Kap 11.3 Lærdom til neste gang

*Lære rammeverket bedre før vi starter prosjektet, da er vi enda mer gardert for å ikke miste arbeid som vi desverre gjorde under en git fuck up.*

Her kunne vi brukt mer tid i startfasen på å lære oss rammeverket vi skulle begynne å bruke litt bedre. Brukte mange timer på å lære det i starten, også kom vi inn i en fase der vi brukte “learning by doing” prinsippet. Det funket bra i starten og litt videre, helt til vi støttet på et git problem som gjorde at vi måtte kjøre rollback en uke tilbake i tid og mistet verdifult arbeid som måtte gjøres om igjen. Dette har påvirket at vi ikke ble ferdig så tidlig som vi skulle ønske. Så mot slutten lærte vi oss skikkelig “branching” så vi ikke skulle få noen problem med ting vi jobbet på selv og selve prosjektet i sin helhet. Vi byttet tidlig over fra trello til icescrum, her brukte vi også “learning by doing” prinsippet som viste seg å gi litt problematikk underveis. Vi lagde i starten av icescrum ikke opp i “stories” men hadde bare en hovedtask på hvert emne og hva som skulle gjøres. Dette viste seg å være vanskelig å estimere timer opp i store bolker, men utover prosjektet ble vi bedre. Vi begynte å dele opp i stories og lagde bolker med hovedemner og til slutt i tasks som vi estimerte timer i. Dette viste seg å være mye enklere og mye mer overskikkelig enn det vi hadde gjort tidligere. Man kan ikke lære seg rammeverket 100% uten og bruke det, men vi skulle brukt litt mer tid på det så hadde vi kanskje spart oss de ekstra timene vi brukte på å rette opp git. Samtidig som å gjøre opp arbeid vi allerede hadde gjort. Man ser også i slutten av prosjektet at teamet har kommet langt iforhold til scrum metoden og dens rammeverk rundt scrum. Vi har alle lært icescrum bedre, og kan si at vi føler oss trygg på å kunne bruke dette videre i en annen sammenheng.

*Lært mer om(icescrum), og fått erfaring med estimater*

*Viktigheten av å møte ofte*

Vi hadde 2 scrum møter hver eneste uke i hele prosjektet, dette fungerte helt ok. Det som kanskje var noen av problemene vi møtte på, var at vi fikk store diskusjoner som egentlig kunne vært plassert i andre møter. Dette gikk utover scrum møte, men så hadde vi ikke scrum møter på 15-20 min hver dag. Vi satte av 2 timer til scrum møter, og i disse ingår diskusjoner og estimeringer mm. For hovedgrunnen med scrum møter er å si hvordan man ligger an, iforhold til det man har gjort og det man skal gjøre. På disse møtene skal man egentlig ikke diskutere problemer, men vi hadde satt av såpass tid at vi gjorde det også. Med andre ord, det er viktig å forstå bruken av scrum møter. Man skal ikke oppnå noe objektivt, men si ifra hvordan man ligger an iforhold til det man holdet på med.

*Gui*

* Vi antok at Qt-jambi ville støtte visning av alle kart. Dette viste seg å være mye mer komplekst enn vi trodde, og resulterte i at programmet vårt kun fikk kjøre på 32-bits java i windows. Dersom vi hadde visst dette på forhånd kunne det påvirket valg av API for GUI.
* Qt-jambi har mye funksjonalitet og fleksibilitet. Dette betyr også lengre opplæring. For prosjektets tidsramme ble det derfor noe tyngre enn hva vi var komfortable med. Her kunne det altså vært bedre med et enklere API. Samtidig er vi fornøyde med å ha lært oss en teknologi som kan anvendes flere ganger senere.
* Mye tid ble brukt til å kode en mulig løsning til grensesnitt, som så ble kastet bort. Grunnen var at klassene som var brukt ikke var egnet for det de ble brukt til. Dette kunne ha vært ungått ved at programmør hadde gjort en grundigere jobb med research før kodingen begynte.
* Økt forståelse for kompleksitet i bygging av grafiske grensesnitt. Vi har også mer forståelse for hva slags komponenter som brukes i dynamiske grensesnitt.
* Det kunne tidligere ha vært et fokus på interoperabilitet mellom GUI og andre komponenter. Dette kunne ha vært gjort gjennom interfaces eller andre former for kontrakter. På denne måten hadde det vært enklere å skrive kode som fungerte med andre komponenter.
* Fra bunnen var det enklest å operere med filstrukturer der alt var samlet i én kategori. Dette ble rotete etterhvert som prosjektet vokste. Dersom det tidligere hadde vært et fokus på mappestruktur ville vi fått et system som var lettere å sette seg inn i for andre.
* Korrekte metoder for debugging av software og god bruk av IDE (eclipse) har hjulpet mye for å kunne kode raskere. Opplæringstid brukt til dette har vært nyttig.
* For å kunne lage løsninger med høy kompleksitet har det i enkelte tilfeller lønnet seg å se på kildekode til andre programmer. Teknikken med å sette seg inn i andre sin kode og gjøre bruk av andres teknikker i egen programvare har vi blitt flinkere til etter å ha kodet GUI.

Å forstå hvem målgruppen for appkliasjonen er, og lage et grensenitt ut ifra dette.

Men går ikke dette litt inn på arkitektur  
**Brukermanual (teknisk og ikke-teknisk)**  
**Teknisk? Whatfuck?**  
  
**Vedlegg-oversikt**

**VEDLEGG**

Teknisk brukermanual  
  
Vi har som sagt tidligere brukt rammeverket qt jambi, dette medfører at vårt programm ikke er plattformuavhengig.  
  
For å kjøre programmet vårt trenger man java 32 bit, windows 7 32/64 bit eller windows 8 64 bit. Det vi har vært så heldig å få bruke er serveren til Lars Erik, så man er ikke avhengig av å være logget på ntnu nettet. Man kan kjøre programmet på hvilket nettverk man vil, og det vil fortsatt fungere.  
**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  
**Foreløpig utelatt (deler av teksten som har blitt tatt ut, men som andre kan ta inn igjen eller jobbe utifra hvis de vil):**  
  
Studentene skal også få en innføring i javaprogrammering i gruppen og hvordan dette løses, ved bruk av versjonskontroll. Hovedpoenget med it1901 faget blir da å få praktiske ferdigheter i forhold til javaprogrammering og ta besluttninger i fellesskap som en gruppe.  
  
  
  
*/\*Putter det her enn så lenge. Kanskje det passer inn, jeg vet ikke, men da burde vi putte inn denne slags info om alle eller de fleste programmene vi har brukt (?)\*/*  
Git er Engelsk slang for “en dum og ubehagelig person” og er utviklet av Linus Torvalds, som også står bak Linux.  
  
  
  
  
  
MYE HER ER BULLSHIT  
Noe av det første vi gjorde var å kartlegge hvor våres individuelle kunnskap lå.   
Tanken var at vi skulle få overordnet ansvar innenfor de ulike delene av prosjektet.  
De ulike ansvarsområdene ble:  
 - Databser  
 - Grafisk grensesnitt  
 - Dokumentasjon  
 -   
Det ansvarsområdet man fikk betydde ikke at man hadde ansvar for å lage den delen av prosjektet alene, det var heller der de personene måtte passe på å ha oversikt OMFORMULER  
  
Resten av den faktiske arbeidsfordelingen ble førs  
  
  
  
Lars Erik kunne ikke komme på første møte, som vi avtalte på kort varsel, noe han gjorde mer enn opp for siden.

**Kap 5.1 Hva slags risiko står man ovenfor i dette prosjektet?**

Det er masse som kan gå galt i et prosjekt som dette, men noen “farer” er viktigere å tenke på enn andre, deriblant kommunikasjonsfeil, sykdom, at noen faller fra og misting av arbeid. Kanskje er det også slik at splitting av gruppen noe man burde vurdere som en viktig risiko, men vi oppdaget ingen tilløp til konflikt.  
  
  
  
Rapportinnhold utifra fagside.  
  
Kap. 1 Introduksjon

Kap 1.1 Om faget, om oppgaven og om gruppen

Kap 1.2 Beksrivelse av prosjektet

Kap 1.3 Valg av prosessmetode (beskrivelse)

Kap. 2 Tidsestimering (sprint besrkivelse, WBS, o.l.)  
Kap. 3 Prosjektorganisering

Kap 3.1 Ansvarsområder

Kap 3.2 Om de forskjellige medlemenes kunnskapsområder

Kap. 4 Arkitektur forklaring  
Kap. 5 Risiko i prosjektet

Kap 5.1 Hva slags risiko står man ovenfor i dette prosjektet?

Kap 5.2 Splitting of the group

Kap 5.3 Uforutsette utfordringer (ny)

Kap 5.4  Deadlines (at man ikke rekker det)

Kap 5.5 Medlemer som ikke møter

Kap 5.6 Kommunikasjon

Kap 5.7 Prosjektplanen failer

Kap. 6 Produktbeskrivelse  
Kap. 7 Prosessbeskrivelse

Kap 7.1 Sprint

Kap 7.2 Arb.fordeling

Kap 7.3 Møter

Kap 7.4 Kildekodedeling

Kap 7.5 Facebook

Kap. 8 System design  
Kap. 9 Implementasjon  
Kap. 10 Tools  
Kap. 11 Evaluering  
 Kap 11.1 Hva gikk bra  
 Kap 11.2 Hva gikk dårlig  
 Kap 11.4 Lærdom til neste gang  
User manual (teknisk og ikke-teknisk)  
Vedlegg: diagrammer, timelister, kildekode, møtereferat, testing  
  
**why qt jambi?**: <http://www.sra.co.jp/qt/relation/qtjambi-whitepaper-tp3.pdf>  
  
  
  
  
Da disse individene ikke hadde noe imot å lære bort, har resten sittet og konsentrert seg , kommet med spørsmål og vist god insattsvilje til å lære.  
  
  
De første møtene var vi stort sett sammen, da vi skulle bli enige om hvordan vi ville at systemet skulle se ut.  
  
  
Dette var et veldig enkelt og grei måte å ha icescrum på egen og pålitelig server. Her logget vi inn og oppdaterte chartene og tasks vi gjorde underveis. Dette har gått veldig bra og smertefritt.  
  
Hele teamet har vært med under hele prossesen, og fått med seg alt som man skulle få med seg. Men alt i alt har vi fungert bra som en gruppe, og ingen har klaget eller gjort seg noen misnøye med at alle ikke endte opp med å gjøre ”like mye” som alle andre.  
  
  
Da var vi så smarte etter en kort tid å bytte til icescrum, som ble det ultimate rammeverket som dekket våre behov iforhold til at vi brukte scrum-metoden som arbeidsmetode. Her fikk vi generert burndowncharts(og div) automatisk. Men i starten her var vi ganske nybegynnere og gjorde en del feil. Vi delte først ikke opp i stories og lagde task utifra de så hver enkelt kunne ta en task. Vi delte opp i store bolker og skrev ned timer på details på hver bolk. Da vi lærte icescrum senere under prosjektet så vi hvor stor nytte det hadde å gjøre det riktig og ordentlig.

**10.1 JavaDoc**

JavaDoc er blitt brukt for dokumentasjon av koden. //Her må noen skrive mer.

**10.2 ER-modeller**

ER-modeller er en abstrakt modell for å modellere databaser (man viser hva slags data tabellene inneholder og hvordan de forholder seg til hverandre). Vi hadde alle lært om ER-modeller i *Informatikk Grunnkurs*, og Lars Erik hadde også hatt et kurs i Databaser hvor ER-modeller var en del av pensum.

**10.3 Dia**

Design av diagrammer er gjort i det GTK+-diagram-baserte programmet Dia. Dette var et greit program som har innebygde funksjoner som er beregnet på bl.a. UML-diagrammer. Ingen utenom Lars hadde noen erfaring med dette programmet, men ble fort kjent med det. Grunnen til at vi valgte dette var at Lars hadde erfaring med programmet fra et annet fag og at det fungerer på både Mac og Windows.

**10.4 UML**

Unified Modeling Language er en standard for IT-relatert modellering innenfor objektorientert programvare. Diverse diagrammer er UML-diagrammer. UML-diagrammer brukes til å klargjøre og gi oversikt over diverse aspekter av hvordan et program skal fungere. Vi har laget UML-diagrammer i planleggingen av prosjektet, og en del av dem har kommet til nytte for å få en generell forståelse av hvordan det ferdige produktet skulle se ut og fungere, og for å ha en felles forståelse av hva vi skal gjøre.

**10.5 Eclipse**

Eclipse er et program for å skrive software (software development environment). Eclipse kan brukes til å skrive i flere språk med ved hjelp av plugins, men har Java som default. Eclipse brukes i kurset *Objektorientert Programmering (TDT4100)*, som alle informatikk-studenter har før de har dette faget.

**10.6 Facebook**

Vi sjekket om faglærer hadde Facebook-profil og så at han hadde det, og regner med at studasser og undasser også har det, så vi antar at alle som leser denne rapporten *ihvertfall* forstår grunnprinsippene. Men for sikekrhets skyld: Viktig å forstå er at man kan være medlemmer av *grupper* og at man på disse gruppene har en felles *wall* der man kan skrive poster og skrive *kommentarer* i respons til poster (på samme måte som wallen til en brukers profil før man fikk timeline). Når det kommer nye poster i en gruppe man følger med på, eller når noen kommenterer på en post du har postet eller kommentert på, eller som du er *tagget* i, får du *notifications*. Vi har valgt å også nevne relevant funksjonalitet med henhold til Facebook-grupper i teksten (kap 7.5).

**10.7 Git**

Git er et system som er laget for folk som jobber på felles IT-prosjekter. Man har en felles prosjektmappe hvor filene er lagret, og systemet håndterer at forskjellige gruppemedlemmer forandrer på filer i prosjektmappa. Hvis forskjellige gruppemedlemmer forandrer på samme fil samtidig får man utfordringer, me også dette er mulig å håndtere.  
  
Hvis man ikke brancher betyr det at man kun har en prosjektmappe som man jobber på, men hvis man brancher betyr det at man jobber på to uavhengige prosjektmapper/”grener” med felles utgangspunkt, hvor en forandring på den ene ikke påvirker den andre. Slik kan man f.eks. lagre programmet som virker i en “gren” og jobbe i en annen. Etterhvert kan man føre grener sammen igjen. //Se gjerne på dette og se om dere synes det er ok.

**10.8 Qt Jambi**

Qt Jambi er et Java bibliotek for å lage grafiske brukergrensesnitt med bindinger mot C++ biblioteket Qt. //Her må noen andre gjerne skrive mer hvis de har noe å føye til.

**10.9 Kartverkets WMS-tjeneste**

//Her må noen skrive mer.

**10.10 Google maps API**

//Her må noen skrive mer.

**10.11 Trello**

Trello er en nettside som hjelper folk med å få oversikt over og koordinere prosjektene sine. Du har lister (som kan representere kategorier av diverse slag) med kort (som kan representere arbeidsoppgaver) som du kan dra rundt omkring. For disse kortene er det diverse funksjonalitet. Du kan f.eks. adde deg selv til et kort for å vise at du jobber på arbeidsoppgaven det representerer, eller flytte kortet til ferdig-lista hvis jobben er gjort.

**10.12 IceScrum**

IceScrum er laget for metodikker som bl.a. Scrum. Digital sprinttavle og burndownchart er innebygde funksjoner i IceScrum, og burndownchart oppdateres automatisk med henhold til sprinttavla.

**10. 13 MySql**

//Her må noen skrive mer.

**10.14 Google docs**

Rapportskrivingen og i forbredelsen av midtveispresentasjonen, samt i planleggingen av enkelte av sprintene, har vi brukt Google Docs. Dette har veldig praktisk siden dokumentene lagres automatisk, og man kan skrive og redigere på det samme dokumentet samtidig, og se hva de andre gjør i real-time.

1. WBS er en måte å visuelt representere alle oppgaver i et prosjekt, laget for å gi økt oversikt. [↑](#footnote-ref-1)
2. Dette er ikke bare noe faglærer advarte mot, men også noe flere av oss kjenner til fra før av erfaring, og noe som som er blitt forsket på. Se f.eks. artikkel om Planning fallacy på Wikipedia: <http://en.wikipedia.org/wiki/Planning_fallacy> (sist sjekket av oss 15.11.12). [↑](#footnote-ref-2)
3. “Branching” i Git: Flere kan jobbe på forskjellige grener/versjoner av prosjektet, og ved å gjøre en “merge” vil disse bli satt sammen til en gren. Det er vanlig å ha en gren (master) som inneholder en stabil versjon av koden. [↑](#footnote-ref-3)
4. JXMapViewer er et bibliotek som skal kunne ta seg av WMS-spørringer og vise dette som et kart i en widget(hva gjør vi her??) i programmet vårt [↑](#footnote-ref-4)
5. Mange av møtene våre var arbeidsmøter hvor mesteparten av tiden uansett gikk til programmering og annet arbeid, men vi brukte litt tid på å planlegge sprinter. [↑](#footnote-ref-5)
6. Se boken "Software Project Management" av Bob Hughes eller denne Wikipedia-artikkelen: http://en.wikipedia.org/wiki/Parkinson's\_law (sist sjekket av oss 14.11.12). [↑](#footnote-ref-6)