IT1901

Grupperapport Gruppe 18

Thomas Almestad - 476701
Nicholas Vallipuram Jaunsen - 476689
Erling Storaker Moen - 475611
Jimmy Pettersen - 476735
Andreas Risvaag - 757763
Tord Standnes - 756451
Endre Michael Austad Ulberg - 476687

Antall ord i rapporten: 5961

Innhold

1 Sammendrag

Målet med kurset var å forbedre våre praktiske evner i gruppearbeid og programmering, ved å bruke arbeidsprosessen Scrum. Vi tilpasset den smidige arbeidprosessen etter gruppens preferanser og meninger om hva som ville funke for oss. Hver sprint lot vi ha en periode på 2 uker og la dermed en plan for å utføre 4 sprinter totalt. Vi planla at den første sprinten skulle brukes for å legge grunnarbeidet slik som oppsett av database osv for å kunne implementere brukerhistoriene. De tre neste tenkte vi å bruke til å implementere brukerhistorier.

Det ble tidlig kartlagt evnesettet til gruppemedlemmene slik at vi kunne fordele ansvar og oppgaver slik som f.eks backend development eller rapportskriving. Hvilken kunnskap vi måtte tilegne oss for å fullføre prosjektet ble også kartlagt, samt hvilke verktøy og programmeringspråk som skulle brukes og en plan for gruppemøter og organisering.

Vi hadde møte med kunde (studass) ukentlig hvor vi fikk svar på hvilke prioriteringer og preferanser han hadde og tilbakemeldinger på produktet. Vi hadde kanskje en i-overkant teknisk orientert testplan hvor den eneste brukertestingen var en akseptansetest med kunde. Resten av vår testplan besto av unittesting, integrasjonstesting og systemtesting.

Siden faget til syvende og sist handler om arbeidsprosess og gruppearbeid, så valgte vi å dokumentere de forskjellige møtene og fremdriften på prosjektet nøye. Vi har skrevet referater fra møtene og har brukt Trello som Scrum board og laget burndown charts manuelt i Google sheets.

Gruppen har hatt en del problemer med ting som ubesluttsomhet, flaskehalser, dårlig oppmøte og kommunikasjon. hvilket tyder på svakheter i arbeidsprosessen og den demokratiske organisasjonen vi har valgt samt lite erfaring med gruppearbeid. Utover i prosjektet ble noen av disse problemene identifisert og forbedret. Prosjektet ble en verdifull erfaring både i gruppearbeid, arbeidsprosess og generell systemutvikling, og til tross for at disiplinen ikke var på topp har alle gruppemedlemmene bidratt mye i prosjektet.

2 Introduksjon

Denne seksjonen gir grunnleggende informasjon om faget, prosjektet, gruppen og arbeidsprosessen.

2.1 Fag

Faget ifølge fagsiden:

> I dette kurset skal en gruppe på 5-9 studenter gjennomføre et mellomstort programmeringsprosjekt. Hovedmålet er å gi studentene forståelse for samspillet mellom prosess og produkt-orienterte utfordringer og aktiviteter i et programmeringsprosjekt. Viktige aspekter er kunnskap om metoder og teorier for organisering av gruppeprogrammeringsprosjekter. Studenter må arbeide med og reflektere over integrasjonen av ulike komponenter for å sette sammen et større programvareprodukt. Oppsummering: Målet med kurset er å forbedre studentenes praktiske ferdigheter i gruppearbeid og i programmering.

2.2 Prosjektet

Oppgaven for dette mellomstore prosjektet er å lage et system for konsertgjennomføring for studentfestivaler. Vi behandlet prosjektet som om det var et ekte oppdrag fra en ekte kunde med tanke på utvikling og testing, men holdt det likevel i bakhodet at arbeidsprosessen var det viktigste.

Oppgaven er oppsummert slik på Blackboard:

> Tenk deg at du skal lage et system for en gjeng i en studentfestival som har ansvar for å arrangere konserter på flere scener, som involverer rigging av scene, teknisk gjennomføring av konserter, samt booking av band.

Kunden skal ha et system som tillater booking, organisering og kontroll av flere scener. I dette skal det være muligheter til å se tilbud fra band, booke band og se hvem som har ansvar for arbeidsoppgaver under forskjellige konserter. Som prosjektleder skulle man få full oversikt over tilbud, hvem som har konsert på hvilke scener og hvem som er ansvarlig under forskjellige show. Det er videre en mengde brukerhistorier som tilspisser krav til funksjonalitet som skulle være en del av produktet.

2.3 Gruppe

Vi er en gruppe på 7 med varierende kunnskapsnivå. Alle på gruppen hadde på starten av prosjektet i det minste litt erfaring med javascript, CSS og HTML mens noen få hadde erfaring med nodeJS, templating engines, og SQL databaser.

Ambisjonen for gruppen til å begynne med var å få A i faget og å lære om gruppearbeid/arbeidsprosess og forbedre individuelle tekniske ferdigheter.

2.4 Arbeidsprosess

Vi brukte det smidige rammeverket scrum for å skape vår egen arbeidsprosess. Hvorfor scrum?

- Iterativ og smidig
- Brukes mye av profesjonelle systemutviklere i næringslivet
- Fremmer gjennomsiktighet og oversikt over status på prosjektet gjennom faste møter

3 Organisering

Fleksibilitet når det kommer til arbeidsoppgaver og veien prosjektet går er en viktig del av Scrum. Derfor har gruppen gjort det slik at alle får prøve seg på det de selv ønsker å jobbe med, og at alle er oppdatert på hva de andre i gruppen jobber med gjennom standup møter.

Vi bestemte oss tidlig for å fordele hovedansvar for forskjellige oppgaver slik at vi fikk litt bedre oversikt over hvem som jobbet med hva, men dette betydde ikke at denne person hadde eksklusivt ansvaret. Til syvende og sist hadde hele gruppen felles ansvar for alle arbeidsoppgavene.

Ved å bruke Trello var det mulig for alle å enkelt se hva de andre jobbet med til enhver tid. Trello gav oss også en veldig oversiktlig måte å følge med på arbeidsprosessen og utviklingen av prosjektet.

3.1 Ansvarsroller

Ansvarsfordelingen var som følger:

Scrum master
Oppgave: Tilrettelegge for en god Scrum-prosess.
Personer: Rotasjon - Alle

Full stack
Oppgave: Utviklere som jobber med å utvikle både frontend og backend av applikasjon
Personer: Nicholas, Jimmy

Frontend

Oppgave: Utviklere som jobber med å utvikle frontend av applikasjon
Personer: Thomas, Tord

Database
Oppgave: Sette opp databasen og generere testdata
Personer: Andreas

Design
Oppgave: Sørge for en helhetlig designprofil. Deriblant fargepalleter og alt annet frontend skal bruke til utforming av siden.
Personer: Erling

Testing
Oppgave: Kjøre systemtest etter at enkeltpersoner i gruppa har unit og integrasjonstestet egne deler og moduler
Personer: Andreas

Dokumentasjon
Oppgave: Sørge for at all kode vi skriver blir fortløpende og skikkelig dokumentert.
Personer: Alle har et ansvar for å dokumentere sin egen kode.

Rapportskriving
Oppgave: Hovedansvar for skriving av rapport
Personer: I starten begynte Endre og deretter Erling, men etterhvert skrev hele gruppen sammen på rapporten.

Vi har fordelt oppgavene slik at de som har ønsket å jobbe med noe har fått lov til det samtidig som man kunne prøvde seg på noe nytt og ukjent. Dette for å bruke prosjektet til å lære mest mulig om de ulike delene av et sammensatt prosjekt.

3.2 Verktøy

Vi har stort sett tatt i bruk verktøy og teknologier som bygger på andre teknologier som gruppen var kjent med fra før. Dette gir oss muligheten til å lære nye og interessante teknologier samtidig som vi ikke trenger å starte helt fra bunnen. Vi har også tatt hensyn til å velge programmer som er gratis, for å gjøre dem tilgjengelig for alle og passet på at de har god dokumentasjon slik at det går raskere å lære seg disse verktøyene. Det siste vi tok hensyn til var å ikke ta i bruk altfor nye verktøy. Med dette mener vi ting som fortsatt er i alpha eller beta stadiene.

3.2.1 Spesifikt

Trello for organisering For organisering og planlegging av oppgaver brukte vi Trello. Noen i gruppen var kjent med det fra før av og mente det ville være en god løsning. Det fantes mer avanserte og utviklede verktøy som Jira, men det hadde blitt for avansert et lite prosjekt som dette. Med Trello var det lett å holde styr på hvilke oppgaver som skulle gjøres i hver sprint og hvem som arbeidet med dem. På Trelloboardet gikk oppgaver fra TODO til Work in progress til Testing og til slutt Done.

Versjonskontroll med Git Vi valgte å bruke git til versjonskontrollering fordi dette er en teknologi som alle på gruppen var kjent med. Dette var også en klar anbefaling fra fagstaben. I utgangspunktet var det meningen at vi skulle bruke NTNU sin egen oppbevaringssted (Stash server), men siden dette tok lang tid å sette opp fikk vi beskjed av fagstaben om å bruke github istedet.

MySQL-Server Vi benyttet oss av NTNU sin serverløsning med MySQL. Det var ikke mange på gruppen som hadde brukt databaser før, og de som hadde erfaring hadde brukt nettopp MySQL. De tok seg da av dette i starten, men etter første uken kunne de fleste på gruppen hente og legge til informasjon fra databasen. Ulempen med MySQL er at man må være koblet til NTNU sitt nettverk via Wi-Fi eller VPN for å kunne bruke programmet vårt.

Discord for kommunikasjon Vi brukte Discord for kommunikasjon i gruppen ettersom de fleste var kjent med dette programmet eller hadde brukt noe tilsvarende før. Discord tillater en enkel og ryddig måte å kommunisere over pc/mobil og deling av dokumenter. I tillegg er det mulig å ha VoIP samtaler over Discord som gjorde det mulig for oss å ha møter over nettet.

Google drive For deling og organisering av dokumenter brukte vi Google Drive ettersom de har en enkel, ryddig og gratis måte å dele dokumenter på hvor man også kan skrive på dokumenter samtidig i sanntid. De har også støtte for regneark, presentasjoner og andre nyttige funksjoner.

Adobe illustrator I starten av prosjektet brukte gruppens designer Illustrator for å lage enkle mockups av hvordan designet skulle bli. Dette programmet støtter blant annet vektorgrafikk som er veldig nyttig for å støtte alle mulige skjermformater av bilder og elementer til nettsiden. I tillegg var flere i gruppen kjent med programmet fra før av.

Node For å gjøre det mulig å skrive serverkode eller "backend" til applikasjonen vår valgte vi å bruke node.js som utvider Javascript slik at det er mulig å skrive serverkode i samme språk som frontend kode, istedet for å bruke et nytt språk, noe som gjør det lettere å lære. I tillegg kommer node med et pakkesystem som heter NPM (Node Package Manager), dette brukte vi til å organisere og holde styr på alle de eksterne bibliotekene som vi bruker i programmet vårt.

Electron Vi bestemte oss for å bruke Electron for å lage en applikasjon fordi oppgaven egnet seg godt til dette. I tillegg var det en god mulighet til å lære nye teknologier samtidig som vi kunne bruke kjente webutviklingsverktøy som HTML, CSS og Javascript. I tillegg til dette er Electron godt egnet for å kjøre på mange forskjellige typer operativsystemer fordi den bruker de samme teknologiene som nettsider bruker og disse støttes universelt, noe som gjør programmet vårt mer tilgjengelig for brukere.

Pug I utgangspunktet hadde vi tenkt å bruke ren HTML for å lage alle sidene i programmet vårt, men vi så raskt at dette ble veldig uoversiktlig og det tok lang tid å skrive alle HTML sidene. Derfor bestemte vi oss for å bruke pug som gjorde det raskere å lage sidene slik at det ble mer oversiktlig og gjorde det mulig å lage maler for sider slik at det ble mer oversiktlig.

Bruk av utviklingsmiljø Siden vi ikke hadde noe kode som var spesifikt til et utviklingsmiljø kunne vi lett dele kildekoden mellom hverandre uten å tenke på hvilket utviklingsmiljø alle på gruppen brukte. Alle på gruppen valgte det utviklingsmiljø de foretrakk mest og var mest kjent med. Da vi spurte alle på gruppen om hvilket utviklingsmiljø de brukte fikk vi disse svarene: VS Code, Atom og WebStorm.

4 Arbeidsprosess

Her utdyper vi om vår implementasjon av Scrum og generelt om hvordan vi organiserte arbeidet.

4.1 Scrum

Scrum er et rammeverk for smidig programutvikling som hvor man har stort sett autonome teams på 3-9 personer. Arbeidet med prosjektet deles opp i oppgaver som kan utføres innen faste tidssykler som kalles sprints. Framdriften loggføres og man holder daglige stand-up møter hvor teammedlemmene besvarer tre sprøsmål:

- Hva har du gjort siden sist?
- Hvilke hindringer/utfordringer har du støtt på?
- Hva har du planlagt å gjøre til neste gang?

Siden dette er et deltidsprosjekt bestemte vi oss for å ha stand-up møter 2 ganger i uken i stedet for daglig og vi lot rollen som Scrum master gå på rotasjon. Etter hver sprint skal fungerende software leveres slik at man kan få tilbakemeldinger og foreta reevalueringer. Scrum er derfor inkrementell og iterativ i kontrast til f.eks fossefallsmetoden hvor kravspesifikasjonen som blir gitt ved start blir fulgt

gjennom hele veien og et fungerende produkt blir kun vist frem til slutt.

I Scrum er smidighet sentralt og er noe vi har holdt fokus på. Smidighet handler om at teamet skal være selvstendig nok til å ta avgjørelser underveis i prosjektet på eget initiativ ettersom de får nye tilbakemeldinger fra brukere, nye kravspesifikasjoner fra kunde eller at andre omstendigheter endrer seg.

Før hver sprint hadde vi et sprint planleggings møte hvor vi bestemte omfanget på den nye sprinten og valgte hvilke brukerhistorier fra backloggen som skulle inkluderes. Brukerhistoriene ble splittet opp i mindre oppgaver som ble gitt hvert sitt tidsestimat.

Etter hver sprint hadde vi et sprint retroperspektiv møte hvor vi analyserte hvordan arbeidsprosessen hadde fungert foregående sprint og kom fram til hva som fungerte bra og hva som kunne forbedres.

4.2 Risikoanalyse

Når man jobber med et prosjekt er det alltid fare for at en del ting kan gå galt. Det varierer når og hvor sannsynlig det er at forskjellige ting skjer, og omfanget. For å visualisere dette bruker vi en risikomatrise, se vedlegg D.1

4.3 Arbeidsfordeling

Med tanke på at Scrum er fleksibelt så har vi prøvd å spre alle arbeidsfelt på alle i gruppen, dermed har alle fått muligheten til å jobbe med de forskjellige delene av prosjektet. Dette har tillatt alle medlemmene på gruppen å få et felles ansvar for fremdrift, der alle har deltatt med sin individuelle del av prosjektet. Dermed er det ingen som alene står ansvarlig hvis noe skulle ha gått galt. Det første vi gjør på hvert møte er å avklare hva hver enkelt skal jobbe med, og før vi går så avklarer vi hva som skal være løst til neste gang.

4.4 Kildekode

For å kunne jobbe på samme kode-repository har vi brukt versjonskontrollsystemet Git. Vi valgte å bruke GitHub istedenfor NTNU sin Git-klient Stash på grunn av at det ikke kom i stand i tide til prosjektet. Git krever at man skriver en kommentar når man har forandret noe, som dermed tidstempler hvilke endringer som har blitt gjort og når.

Det har ellers vært forskjellige preferanser i bruk av programvare, i hovedsak VS Code eller WebStorm, og begge disse lagrer noen spesifikke filer for lokale innstillinger som ikke kan brukes andre steder. Vi spesifiserte derfor i .gitignore filen vår at disse filene ikke skulle pushes til git, og ungikk på denne måten en del mindre kompatibilitets-problemer som kunne ha oppstått ved bruk av forskjellige programmer. Vi har vært opptatt av at man skal committe og

pushe ofte, samt at man skal jobbe i egne branches. Ved å jobbe i egne branches fikk vi mer oversikt over hva som ble jobbet på og progresjon i individuelle tasks.

Når vi startet arbeidet på en ny brukerhistorie, lagde vi en ny branch. Når vi var ferdig med brukerhistorien og hadde fullført unit testing, merget vi dette til development branchen. Når brukerhistorien hadde bestått resten av testene merget vi development branchen til master branchen. Det var kun tillatt å merge med master-branchen dersom en var helt sikker på at alle testene var bestått i development branchen. Slik vil alt som blir en del av det endelige produktet være feilfritt.

5 Produkt

Vi bestemte oss for å lage en desktop applikasjon ved hjelp av Electron som er en pakke i nodeJS. Electron benytter seg av webteknologier slik som HTML, CSS og Javascript til å skape et brukergrensesnitt.

5.1 Arkitektur

Flere ulike systemarkitekturer ble vurdert for prosjektet, blant dem var en lagvis systemarkitektur, en model-view-controller(MVC) arkitektur og en "repository" arkitektur.

5.1.1 Repository arkitektur

I en repository arkitektur lagres dataene i systemet i et sentralt lager tilgjengelig for alle eller de fleste komponentene i systemet og all interaksjon mellom komponenter foregår via behandling av dataene i dette lageret.

Det endelige valget av systemarkitektur for prosjektet ble repository arkitektur. Dette ble gjort ettersom beskrivelsen av systemet ga inntrykk av at det ville bli nødvendig å behandle og forholde seg til brukerdata regelmessig og at den samme dataene gjerne kunne vise seg å være nyttig å beholde i lengre perioder. Denne systemarkitekturen kom med fordelen at hver komponent i systemet ble isolert fra hverandre og ikke var avhengige av noen form for kunnskap om hverandre, noe som gjorde arbeidet med å utvikle de forskjellige komponentene i systemet i parallell enklere.

5.1.2 Lagvis arkitektur

En lagvis systemarkitektur er en systemarkitektur der funksjonalitet i systemet organiseres i flere hierarkiske lag der hvert lag består av komponenter med relatert funksjonalitet som tilbyr tjenester til høyere lag.

Det ble bestemt at det ikke ville være nødvendig med en lagvis separasjon av systemet da de fleste av oppgavene i systemet besto av direkte manipulering av

data og at kunden mente at sikkerhet ikke var høyt prioritert i systemet så det var lite behov for å modellere inn lag for autentisering og lignende.

5.1.3 MVC

MVC arkitekturen er basert rundt en tredeling av systemet. Det deles i de logiske komponentene, modeller, views og controllere. Disse komponentene representerer delene av systemet som forholder seg til interaksjon (controller), data (model) og presentasjon av data til bruker (view). I MVC arkitekturen kommuniserer komponentene med hverandre for å danne et helhetlig system der controller oversetter brukerhandlinger til modell oppdateringer og valg av view, view presenterer model grafisk og modell inneholder applikasjons tilstand og informerer view om tilstandsendringer.

Ettersom forventet data model og interaksjoner i systemet kunne modelleres svært enkelt og det dermed var lite å tjene på å separere dem fra presentasjonen av data ble det bestemt at å forsøke å separere koden i de tre logiske komponentene ville øke kompleksiteten i systemet mer enn nødvendig uten å medføre noen merkbare fordeler over repository arkitekturen.

5.2 Sluttresultat

Det følgende er en liste over brukerhistoriene som ble fullstendig eller delvis implementert:

1	Som arrangør av en konsert skal jeg kunne se en liste med hvem som hjelper til med rigging og tekniske oppgaver under konserten.	Ferdig
2	Som arrangør av konserter ønsker jeg å få opp en totaloversikt over alle konserter på alle scener under en festival.	Ferdig
3	Som bookingansvarlig skal jeg kunne sende tilbud til manager for et band på booking på en spesifikk dato og med en spesifikk pris. Tilbudet skal være godkjent av bookingsjef.	Ferdig
4	Som bookingansvarlig så skal jeg kunne søke opp band som har spilt på en scene tidligere, og se nøkkelinformasjon om bandet og om tidligere konserter slik at jeg kan vurdere hvilke band som bør få tilbud om å bli booket.	Ferdig
5	Som bookingansvarlig ønsker jeg å kunne få en oversikt over hvilke konserter som har vært på tidligere festivaler innen en gitt sjanger, med informasjon om publikumsantall og scene for å vurdere kommende konserter innen sjangeren.	Delvis ferdig
6	Som bookingansvarlig ønsker jeg å kunne se nøkkelinformasjon om et band, som popularitet i strømmetjenester, salgstall for album og tidligere konserter i Norge for å vurdere om bandet skal få tilbud om å bli booket.	Ferdig
7	Som bookingansvarlig ønsker jeg å kunne vise en liste med tekniske behov for et band for å forberede de som har teknisk ansvar.	Ferdig
13	Som lyd eller lystekniker skal jeg kunne få opp en oversikt over konserter jeg skal jobbe med.	Ferdig
14	Som manager for et band ønsker jeg å kunne melde inn tekniske behov for et band som skal spille på en scene slik at bevhovene er kommunisert tydelig.	Delvis ferdig

6 Tidsbruk

Man gjør tidsestimater for å finne ut hvor mye arbeid man kan gjøre før deadlinen til prosjektet. Vi brukte releaseplan, brenndiagram, velocity calculation med fokus faktor, storypoints (timer i vårt tilfelle) og timelister for hvert gruppemedlem.

På vårt første møte, gjorde vi vårt første tidsestimat som viste seg å bli grunnleggende for resten av tidsestimatene våre. Estimatet var hvor mye vi skulle jobbe i gjennomsnitt hver uke. Vi kom fram til at 8 timer per uke var et greit estimat sett i kontekst med de andre fagene. Arbeid med faget kan deles inn i omtrent 4 kategorier: lesing av pensum, "møter, administrativt og slakk", programmering og rapportskriving. Vi gjorde et estimat og kom fram til en fordeling:

- Lesing av pensum: 10%
- Møter, administrativt og slakk: 30%
- Programmering: 30%
- Rapportskriving: 30%

Scrum rammeverket omfatter hovedsakelig "møter, administrativt og slakk" og "programmering" kategoriene, derfor er disse beskrevet i detalj i denne rapporten, de andre kategoriene er ikke skildret. Vi må likevel ha det med i tidsestimatene for å få reelle estimat og beregninger.

Vi bruker "Scrum and XP from the Trenches" definisjon av velocity, "Velocity is a measurement of "amount of work done", where each item is weighted in terms of its initial estimate." (Kniberg, 2007, s.25). Her refererer "amount of work done" til programmering kategorien. Utregningen av velocity er: "This sprint's estimated velocity: (available man-days)x(focus factor)=(estimated velocity)" (Kniberg, 2007, s.27). Hvor "focus factor" her blir det samme som programmeringskategorien beskrevet over. Det gir oss (7 personer x 8 timer x 8 uker)x(30%)=(134.4 timer).

6.1 Releaseplan vs burndown

Brenndiagram viser ideelt arbeid utført mot faktisk arbeid utført over tid. Estimatene er basert på tasks i sprinten som igjen er bassert på estimatene av userstories. Releaseplanen er en overordnet oversikt av når userstoriesene skal være ferdig. Releaseplanen bruker kun tidsestimatet til userstorien og bruker ikke tasks som i beregningene sine. Dette er problematisk når en userstory ikke blir ferdig i en sprint og går over i neste. Arbeid som blir gjort i en sprint blir ført i den neste, ettersom alt arbeidet som blir gjort i en sprint blir ført over i den neste. I burndown chartet derimot blir det mer nøyaktig ettersom hver story er delt opp i tasks, så hver fullført task blir ført opp. Vi har valgt å

estimere alt i timer, både burndown charts og releaseplan. Det passer best med tiden vi har tilgjengelig. Estimat i dager er ikke kompatibelt med tanke på at vi kun har 1-2 arbeidsdager per uke som blir veldig unøyaktig og vanskelig å regne med.

6.2 Brenndiagram

Vi har 5 brenndiagram, en for hver sprint og en for hele prosjektet. Ettersom brenndiagramene ble laget for hver sprint samsvarer de med fullførte tasks og ikke fullførte userstories slik som releaseplanen gjør. Vi jobbet hovedsaklig på de dagene vi var samlet, dette kan man se på brenndiagrammene ta man har fått gjort mesteparten av tasksene på et par dager.

6.3 Releseplan

Releaseplanen skal vise hvilke brukerhistorier som skal være ferdig i hver sprint. Vi spurte under de ukentlige møtene med kunden om det var nye prioriteringer. Det ble laget 3 iterasjoner av releaseplanen, oppdatert i forhold til nye brukerhistorier og nye ønsker fra kunde. Vi varierte antall brukerhistorier avhengig av fokus. For hver nye versjon av releaseplanen reestimerte vi tiden til userstoriesene basert på hvor lang tid vi brukte på de forrige. I et ekte arbeidsscenario hadde det holdt med å gjøre disse estimatene en gang, men siden vi er såpass uerfarene med å estimere tidsbrukt så gjorde vi det flere ganger, hver gang med litt mer erfaring.

I releaseplan v1 hadde vi ca. 4 userstories i hver sprint. Dette passet med tidsestimatene vi hadde gjort. Userstoriene som var høyest prioritert var de uferdige fra de tidligere sprintene.

Når vi ble ferdig med sprint 1 oppdaterte vi releaseplanen til versjon to. Det ble påpekt at dette var feil framgangsmåte i følge Scrum rammeverket. Følger man Scrum skal man regne ut sin framtidige fokusfaktor basert på teamet sitt forrige fokusfaktor fra tidligere sprinter (Kniberg, 2007, s.27). Vi hadde ingen userstories fullført og kunne ikke skrive opp tasks som hadde blitt utført i releaseplanen. Så fokuset vårt var 0% i følge releaseplanen, noe som var feil. I ettertid burde vi brukt totalt antall timer utført fra brenndiagramet til sprint 1, til å lage et midlertidig estimat for vårt teams velocity.

I releaseplan v3 hadde vi fullført 2 sprinter, vi hadde den faktiske velocityen til teamet fra de 2 første sprintene, og vi så hvor feil våre estimater hadde vært. På dette stadige gikk ting alt for seint framover, men vi forsto ikke helt hvorfor. Userstoriene var ikke spesielt vanskelige i seg selv og burde ikke ta så lang tid å gjennomføre. Etter litt diskusjon fant vi ut at mye av problemet lå i å bruke stacken vi hadde valgt. Det tok tid å lære seg hvordan ting funket og å finne ut hvor feil lå når de først kom opp. Når vi gjorde et nytt tidsestimat nå så tok vi høyde for tiden det tar å jobbe med et ukjent rammeverk.

Videre så vi at arbeidsmengden i brenndiagramene ikke samsvarer med releaseplan v1 og v2. Dette var fordi vi ikke hadde håndtert tekniske userstories på en forsvarlig måte. Vi hadde bare opprettet dem som tasks i sprinten når de dukket opp, noe som var feil i følge Scrum. I følge scrum skal vi enten gjøre dem om til en userstorie med verdi, inkorperere dem i en annen userstory, eller ha en seperat liste for tekniske userstories (Kniberg, 2007, s.39). Vi valgte å inkorperere de tekniske userstoriene i de andre userstories, fordi vi ikke greide å vinkle de tekniske userstoriene slik at de hadde en verdi for kunden. Vi estimerte ikke tiden på de nye userstoriesene fordi vi så fort at vi ikke kom til å få tid til å gjøre dem uansett.

6.4 Konklusjon

I vedlegg B.2.5 ser vi antall timer det tar å gjennomføre releaseplanene (relativt til den releaseplan v1). Man kan omtrentlig si at antall timer utført samsvarer med timeberegningene i releaseplanene, som egentlig bare betyr at vi har jobbet omtrent så mye som vi har sagt at vi skal gjøre. Det man også kan se ut i fra figuren er at vi har feilberegnet estimatene av userstoriene kraftig ettersom vi har færre og færre userstories med for hver verson av releaseplanen, mens timearbeidet for releaseplanen er cirka det samme.
Oppsummert kan vi si at det har vært mye rot med tidsestimater i startfasen av prosjektet når vi ikke hadde full innsikt i hvordan Scrum rammeverket fungerer, men vi fikk det ordnet opp i det halvveis ut i prosjektet og siden det har vi fått gjort det riktig.

7 Testing

Når man benytter seg av smidig utvikling som arbeidsmetode bør systemet testes fortløpende. Funksjonalitetstester og brukertester må gjøres fortløpende slik at man oppdager feil som må rettes opp eller ting som må reprioriteres. Se også vår testplan i vedlegg D.2.

7.1 Testplan

Vi valgte å teste alle komponenter så fort de er fullført, og teste dem i samspill med de komponentene de skal samarbeide med så fort de også er klare. Resultatene fra testingen blir brukt når vi jobber videre på prosjektet.

Vi jobber i en situasjon der vi ofte møter kunden. Slik at vi ofte får input og kan gjøre fortløpende endringer underveis i henhold til kundens ønsker. Måten vi har delt opp testingen på i to faser gjør det også lett å gjennomføre endringer som ikke var planlagt uten å måtte avvike fra testplanen vår og repetere tester, noe som kan hindre arbeidsflyten til gruppen.

7.2 Unit Testing

Hensikten med å utføre unit tester er å teste de minste komponentene i et system i isolasjon for å bekrefte at de oppfører seg som ønsket. Dette lar utviklerene oppdage å isolere mange feil lettere ettersom komponentene, i den grad det er mulig, testes i isolasjon og feilen dermed kun kan stamme fra et liten del av programmet.

I prosjektet ble unit tester gjennomført fortløpende av utvikleren ansvarlig for en funksjon på komponentene han jobbet med.

Unit tester kan enten utføres manuelt eller automatisk gjennom gjennom unit testing programvarer som Karma for Javascript. Vi diskuterte dette i gruppen og bestemte oss tilslutt for å la være å bruke Karma fordi vi var bekymret for at det ville gå mye tid til å skrive alle testene med tanke på at vi i allerede måtte ha satt av mye tid til å lære oss nye teknologier, bla. Electron og Pug.

7.3 Integrasjonstesting

Etter at unit testing var gjennomført ble det gjort integrasjonstester på større sammenhengende deler av systemet. I integrasjonstestene ble de små isolerte komponentene som besto unit testing kombinert og testet som større enheter. Dette gjøres for å oppdage mulige problemer med arbeidet som er gjennomført og ettersom alle komponentene som slipper gjennom til integrasjonstesting har bestått unit testing er det dermed sannsynlig at problemer som oppdages i denne fasen stammer fra uventede interaksjoner mellom dem.

Integrasjonstesting ble gjennomført fortløpende i development branchen etterhvert som hele komponenter ble fullført for å forsikre at alle komponenter i systemet virket slik de skulle. Disse testene gjorde vi også manuelt.

7.4 Systemtesting

Med jevne mellomrom ble systemtester utført av testansvarlig for prosjektet. Her testes hele systemet med fokus på at systemet oppfyller prosjektkravene. Denne formen for testing skiller seg derfor fra unit- og integrasjonstester som begge er fokusert på å teste intern funksjonalitet i systemet. For å oppnå dette er systemtestene derfor formulert for å forsøke å modellere høynivå beskrivelser av systemet slik som de funksjonelle og ikke-funksjonelle kravene for systemet.

Systemtestene modellerer emergent oppførsel i systemet ettersom de er designet for å modellere interaksjoner mellom komponentene i systemet. Slik kan de brukes både til å identifisere situasjoner der systemet ikke oppfører seg på en ønsket måte, og til å identifisere situasjoner der systemet oppfører seg på en uforventet måte med negativ påvirkning på brukeropplevelsen av systemet.

Ettersom alle deler av systemet testes på et høyere nivå i system testene er det derfor viktig at system testene kun utføres på komponenter som har bestått både unit og integrasjonstester ettersom feil i intern funksjonalitet i systemet under en systemtest kan føre til at problematisk oppførsel i systemet enten ikke oppdages eller oppdages der de ikke eksisterer.

7.5 Akseptansetesting

I de siste møtene med kunden gikk vi gjennom de brukerhistoriene vi hadde fullført og testet ferdig. Kunden fikk dermed akseptansetestet brukerhistoriene og godkjent dem. De brukerhistorien som er med her er altså de vi klarte å få ferdig til kodefrys. Følgende er resultatet av akseptansetestingen.

1	Som arrangør av en konsert skal jeg kunne se en liste med hvem som hjelper til med rigging og tekniske oppgaver under konserten.	Godkjent
2	Som arrangør av konserter ønsker jeg å få opp en totaloversikt over alle konserter på alle scener under en festival.	Godkjent
3	Som bookingansvarlig skal jeg kunne sende tilbud til manager for et band på booking på en spesifikk dato og med en spesifikk pris. Tilbudet skal være godkjent av bookingsjef.	Ikke godkjent
4	Som bookingansvarlig så skal jeg kunne søke opp band som har spilt på en scene tidligere, og se nøkkelinformasjon om bandet og om tidligere konserter slik at jeg kan vurdere hvilke band som bør få tilbud om å bli booket.	Godkjent
5	Som bookingansvarlig ønsker jeg å kunne få en oversikt over hvilke konserter som har vært på tidligere festivaler innen en gitt sjanger, med informasjon om publikumsantall og scene for å vurdere kommende konserter innen sjangeren.	Ikke godkjent
6	Som bookingansvarlig ønsker jeg å kunne se nøkkelinformasjon om et band, som popularitet i strømmetjenester, salgstall for album og tidligere konserter i Norge for å vurdere om bandet skal få tilbud om å bli booket.	Godkjent
7	Som bookingansvarlig ønsker jeg å kunne vise en liste med tekniske behov for et band for å forberede de som har teknisk ansvar.	Godkjent
13	Som lyd eller lystekniker skal jeg kunne få opp en oversikt over konserter jeg skal jobbe med.	Godkjent
14	Som manager for et band ønsker jeg å kunne melde inn tekniske behov for et band som skal spille på en scene slik at bevhovene er kommunisert tydelig.	Ikke godkjent

8 Sprinter

Vi brukte sprinter som er den anbefalte arbeidsmetoden i Scrum. Etter hver sprint skal man kunne levere et fungerende produkt, med nye funksjoner som har blitt laget.

8.1 Lengde og innhold

Avhengig av prosjektet man jobber med må man vurdere hvilken lengde man vil ha på sprintene sine, dette burde være det samme gjennom hele prosjektet.

Vi valgte å ha to uker lange sprinter, og dette viste seg å være en riktig lengde for oss. Dette tillot oss å ha jevnlige oppdateringer samtidig som arbeidstiden ikke ble dominert av møter.

8.2 Organisering av sprintene

Vi valgte å ha gruppemøter på mandager og torsdager, på torsdager var det møte med kunde og veileder. Hver torsdag hadde vi da møte med kunde der vi sjekket om det var omprioriteringer som måtte bli gjort i backlogen, eller andre ønsker som kunden måtte ha. Dette passet godt med tanke på at vi vanligvis hadde sprintplanleggingsmøtet den samme dagen. Et eksempel på et slikt ønske var når kunden gav oss mange nye brukerhistorier å jobbe med. Da brukte vi gruppemøtet til å omprioritere backloggen og lage en ny releaseplan. Deretter startet vi sprintplanleggingsmøtet.

Vi valgte brukerhistoriene for den respektive sprinten slik som det sto i releaseplanen vår. Disse ble da delt inn i tasks, som vi så la på trelloboardet for sprinten. Deretter kunne alle på gruppen velge hvilke tasks de ville gjøre. Når alle taskene under en brukerhistorie var gjennomført og testet var brukerhistorien fullført.

Hver sprint startes med et sprintplanleggningsmøte, og det avsluttes med et retrospekt. Sprintene våre startet på mandager og ble avsluttet torsdager halvannen uke senere. Når vi ikke hadde sprintplanleggingsmøte startet alle møtene våre med stand-up. Stand-upene våre bestod av 3 spørsmål:

- Hva har du gjort siden sist?
- Har du noen utfordringer?
- Hva skal du gjøre til neste gang? ...

9 Retrospekt

Etter hver Sprint periode har vi samlet oss og gått gjennom arbeidet som har blitt gjort. Vi ser tilbake på hva som har gått bra og hva som kunne blitt gjort bedre. Vi har gjennomført møtene etter Scrum modellen for retrospektive møter.

Gruppen oppsummerer sprinten
Scrum master går gjennom backloggen og gruppen går over arbeidet som har blitt gjort i denne sprinten.

Runde rundt bordet

Alle i gruppen får en sjanse til å si hva de synes gikk bra, hva de synes kunne vært annerledes og hvilke tiltak de synes vi skal gjøre for å jobbe bedre i neste sprint.

Estimert tid mot faktisk tid
Gruppen sammenligner mengden arbeid som har blitt gjort opp mot arbeidet vi planla å gjøre og brukte denne informasjonen til å justere fremtidige velocity.

Konklusjon
Scrum master oppsummerer hvilke tiltak vi skal iverksette for å forbedre arbeidsprosessen.

Sprint 1:
I den første sprinten var det en del problemer. I hovedsak dreide dette seg om at vi var dårlige på å kommunisere med hverandre, mange var ikke vant til kommunikasjonsverktøyet, Discord, som vi hadde valgt å bruke. Dette gjorde at det ble lav effektivitet og mange måtte vente på å få opp databasen og designet. Tidsestimeringen ble også da helt feil.

Sprint 2:
Her skulle vi få gjort ganske mye, men det ble gjort relativt lite. Kommunikasjonen bedret seg, som var hovedproblemet fra forrige sprint. Gruppen merket at som en helhet så var det et problem at ikke alle var kjent med hele stacken vi jobbet med. Dette førte til at flere fikk begrensninger i høyt fokus de fikk. Sammen kom vi fram til at det ville være krevende å bytte stack, og valgte derfor heller å senke fokuset og velocity. Tidsestimatene ble bedre, men var fortsatt for optimistiske.

Sprint 3:
I denne sprinten fikk vi bedret kommunikasjon og fikk gjort flere brukerhistorier, men vi var nå klar over våre tidligere opptimistiske estimater om team velocity og justerte den basert på vår team velocity fra tidligere sprinter. Vi innså at vi ikke fikk gjort like mye som vi hadde håpet.

Sprint 4:
I sprint 4 hadde vi endelig team velocity på plass, selv om den var lavere en vi hadde ønsket. Dårlig oppmøte som ble sett på som hovedproblemet i denne sprinten. Kommunikasjonen i gruppen ble også svekket på grunna av dårlig oppmøte, måten vi løste dette på var å ta møtene over Discord i stedet for å møte opp fysisk slik at vi fortsatt fikk kommunisert med hverandre.

Retrospekt konklusjon
For å konkludere så har har det vært en del oppturer og nedturer. Det å ha møter der vi snakker om det som har gått bra, og det som kan forbedres har vært viktig. Det har gitt alle muligheten til å få tilbakemeldinger og hjelp til å forbedre seg samtidig som vi har fått gjennomført konkrete tiltak for å forbedre

gruppesamarbeidet.

Utover i prosjektet ble kommunikasjonen og samarbeidet bedre, og vi klarte å tilpasse oss til hverandre bedre, selv om gruppen slet litt med oppmøte mot slutten av prosjektet.

Gruppen i sin helhet klarte å utnytte flere deler av Scrum rammeverket for å :

- raskt kunne tilpasse oss endringer i kundens ønsker om produktet
- dele opp store oppgaver i mindre "tasks" for å gjøre arbeidet mer håndterlig
- få god oversikt over fremgang i prosjektet og hva andre på gruppen jobber med gjennom stand-up møter og sprint reviews
- forbedre velocity via sprint retrospektiver

Hvis vi skulle gjort denne prosessen på nytt så ville vi nok startet med å teste idéen vår om brukergrensesnitt, valgt en stack der flere kunne alle rammeverkene vi brukte fra før av og satt møtetidene senere på dagen for å få bedre oppmøte eller evt. endret møtetidene når vi begynte å oppleve dårlig oppmøte.

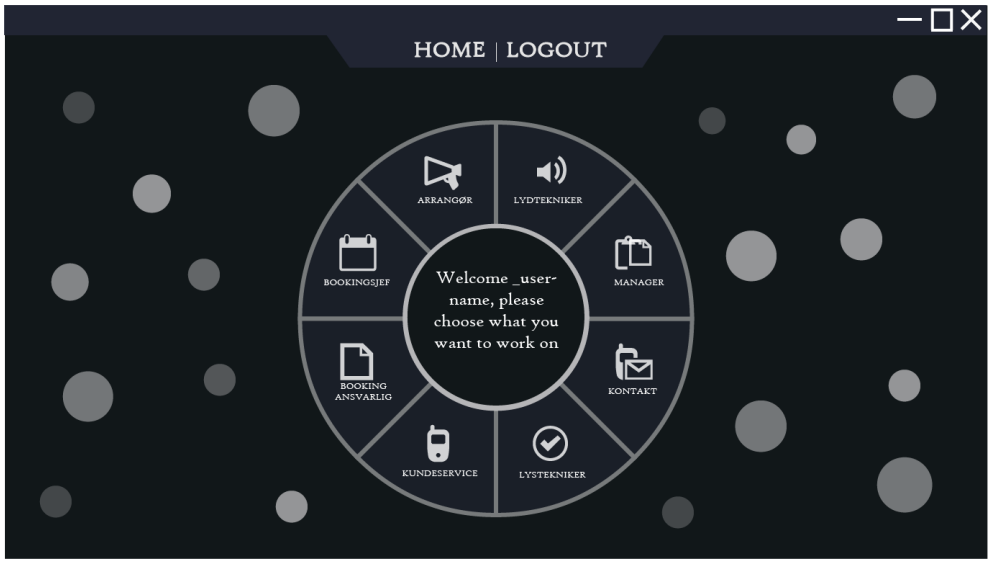
10 Referanseliste

Kniberg, H. (2007). *Scrum and XP from the Trenches.* Hentet fra https://ntnu.blackboard.com/webapps/blackboard/content/listContent.jsp?course_id=_3691_1content_id=_50344_1mode=reset

Slettevold, J. (2017). Veileder, Produkteier.

A Vedlegg - Skisser

A.1 Hovedmeny



A.2 Arrangørside

B Vedlegg - Tidsestimering

B.1 Realeaseplaner

Realeaseplan v1

Nr.	User stories
1	Som arrangør av en konsert skal jeg kunne se en liste med hvem som hjelper til med rigging og tekniske oppgaver under konserten.
2	Som arrangør av konserter ønsker jeg å få opp en totaloversikt over alle konserter på alle scener under en festival.
3	Som bookingansvarlig skal jeg kunne sende tilbud til manager for et band på booking på en spesifikk dato og med en spesifikk pris. Tilbudet skal være godkjent av bookingsjef.
4	Som bookingansvarlig så skal jeg kunne søke opp band som har spilt på en scene tidligere, og se nøkkelinformasjon om bandet og om tidligere konserter slik at jeg kan vurdere hvilke band som bør få tilbud om å bli booket.
5	Som bookingansvarlig ønsker jeg å kunne få en oversikt over hvilke konserter som har vært på tidligere festivaler innen en gitt sjanger, med informasjon om publikumsantall og scene for å vurdere kommende konserter innen sjangeren.
6	Som bookingansvarlig ønsker jeg å kunne se nøkkelinformasjon om et band, som popularitet i strømmetjenester, salgstall for album og tidligere konserter i Norge for å vurdere om bandet skal få tilbud om å bli booket.
7	Som bookingansvarlig ønsker jeg å kunne vise en liste med tekniske behov for et band for å forberede de som har teknisk ansvar.
8	Som bookingsjef skal jeg kunne få en oversikt over antall konsertdatoer booket, antall tilbud sendt og antall ledige datoer.
9	Som bookingsjef skal jeg kunne godkjenne eller avslå tilbud til band som er forberedt av bookingansvarlige.
10	Som bookingsjef skal jeg kunne se en oversikt over bookingtilbud som er sendt til band.

Realeaseplan v2

Team velocity	
Antall personer	7
Timer per uke	8
Antall uker en sprint varer	2
Arbeidstimer tilgjengelig per uke	112

Sprint 1			
Estimert fokus	30%		
Faktisk fokus	0.00%		
Totale arbeidstimer tilgjengelig	33.6		
Estimert tid til overs	33.6		
Story		Tid estimert	Tid brukt
-		0	0
Sum		0	0

Sprint 2			
Estimert fokus	30%		
Faktisk fokus	16.96%		
Totale arbeidstimer tilgjengelig	33.6		
Estimert tid til overs	23.6		
Story		Tid estimert	Tid brukt
1 Tekniske hjelpere		10	6
2 Oversikt scener og konserter		10	13
Sum:		20	19

Releaseplan v3

Team velocity	
Antall personer	7
Timer per uke	8
Antall uker en sprint varer	2
Arbeidstimer tilgjengelig per uke	112

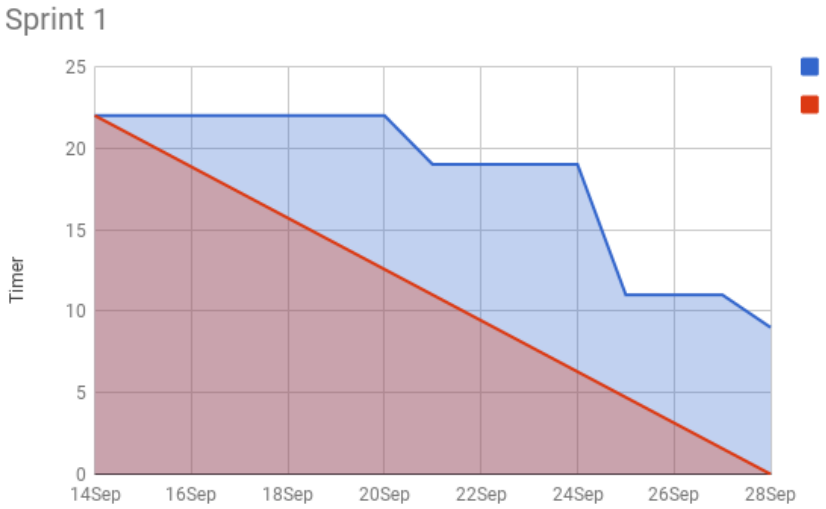
Sprint 1			
Estimert fokus	30%		
Faktisk fokus	0.00%		
Totale arbeidstimer tilgjengelig	33.6		
Estimert tid til overs	33.6		
Story		Tid estimert	Tid brukt
-		0	0
Sum		0	0

Sprint 2					
Estimert fokus	30%				
Faktisk fokus	45.54%				
Totale arbeidstimer tilgjengelig	33.6				
Estimert tid til overs	8.6				
Story		Tid estimert	Tid brukt		
1 Tekniske hjelpere		25	23	login inkorperert i tidsestimatet	
2 Oversikt scener og konserter		25	28	backend inkorperert i tidsestimatet	
Sum:		50	51		

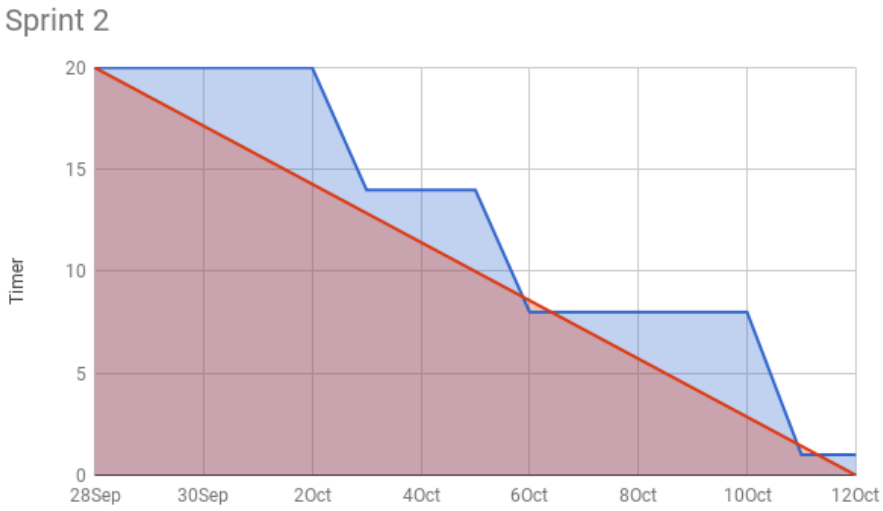
Sprint 3					
Estimert fokus	20%				
Faktisk fokus	28.57%				

B.2 Burndowncharts

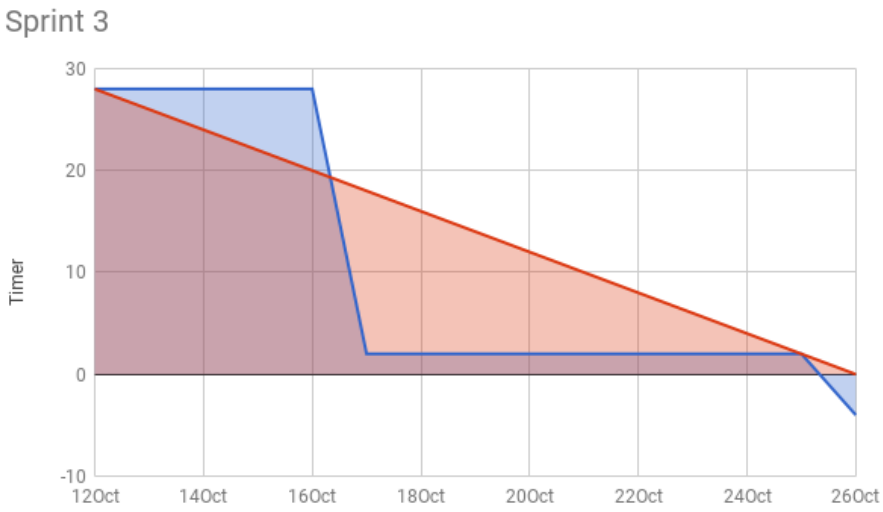
B.2.1



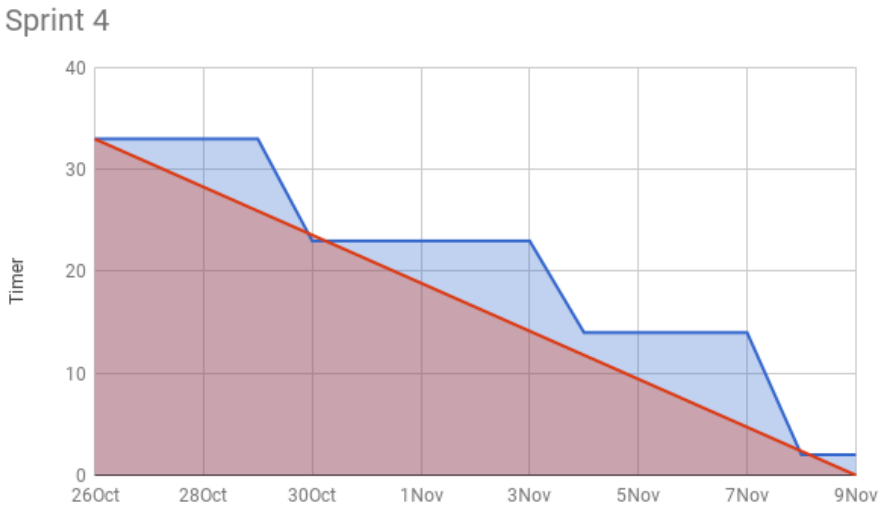
B.2.2



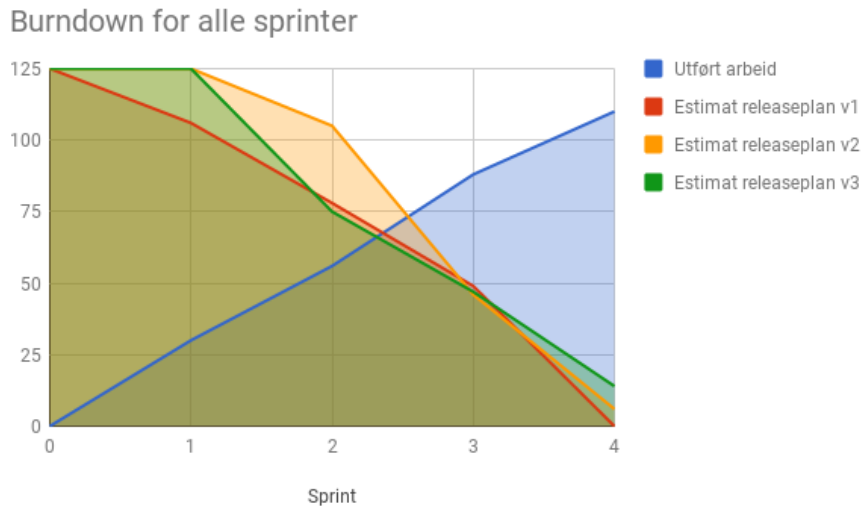
B.2.3


Sprint 3
Timer
30
20
10
0
-10
12Oct
14Oct
16Oct
18Oct
20Oct
22Oct
24Oct
26Oct

B.2.4


Sprint 4
Timer
40
30
20
10
0
26Oct
28Oct
30Oct
1Nov
3Nov
5Nov
7Nov
9Nov

B.2.5



C Vedlegg - Manual

Installasjon av applikasjonen For å installere applikasjonen kan du lese README.md filen som ligger på githubsiden vår (https://github.com/skykanin/Gruppe18/blob/master/READ). Der står det hva som er nødvendig for å installere programmet og hvordan dette skal gjøres.

Filstruktur

- /Gruppe18
- /css
- /html
- /images
  - /login icons
  - /menu icons
- /includes
- /js
- main.js
- /node_modules

- package.json
- package-lock.json
- README.md

Under css mappen ligger alle css-filene for design av applikasjonen. Under html mappen ligger alle pug filene som kompileres til html filer i electron og includes mappen inneholder en mal for disse sidene, for å lese mer om teknologiene vi brukte se kapittelet om verktøy i grupperapporten. Alle bilder og ikoner ligger under images. Alt av logikk for applikasjonen ligger under js mappen. Main.js filen er start filen for electron som starter applikasjonen og inneholder generell logikk for hele programmet. I package.json og package-lock.json filene finner du en liste over alle de eksterne bibliotekene vi bruker i applikasjonen vår og disse filene beskriver alt som installeres i node_modules mappen (her ligger alle bibliotekene vi har lagt til og node sine egne biblioteker). README.md filen beskriver installasjonsprosessen.

Registrering av bruker Når du starter applikasjonen må du registrere en bruker for å få tilgang til applikasjonen. Dette kan gjøres via knappen nederst på login-boksen. Så fyller du inn alle feltene og velger brukertype. For å få tilgang til alle delene av applikasjonen kan du velge brukertype "administrator".

Hjemmeside Etter å ha blitt innlogget kommer du til hjemmesiden hvor du har knapper for alle brukertypene. Øverst i høyre hjørne har du også en avlogging knapp og øverst til venstre har du en hjem knapp som alltid tar deg tilbake til hjemmesiden.

Bookingansvarlig Inne på denne siden kan du søke etter band, resultatet er en liste over band med informasjon om manager, sjanger, en deskripsjon, antall strømninger og informasjon om tidligere opptredener.

For å se informasjon om hvilke tekniske behov et band har, kan man trykke på "band needs" for å komme til en ny søkeside som viser oversikt over bandnavn og tekniske behov som tilhører bandet.

Bookingsjef, kontakt og kundeservice Disse knappene har ingen funksjonalitet fordi vi ikke klarte å fullføre brukerhistoriene som tilhører disse sidene.

Lydtekniker På denne siden kan du finne informasjon over alle de forskjellige lydteknikerne og hvilke konserter de tilhører. Konsertinformasjon viser bandnavn, scene og dato.

Lystekniker Under denne siden finner du oversikten over alle lysteknikerne og hvilke konserter de tilhører. Konsertinformasjon viser bandnavn, scene og dato.

Manager Denne siden lar deg velge et band og legge til tekniske behov til bandet. På høyre siden har du en liste over alle band og et søkefelt for å søke etter spesifikke band. For å velge et band å legge behov til trenger du bare å trykke på bandet i listen, skrive inn informasjonen i tekstfeltet og trykke "submit" knappen NB. Kjent bug i denne delen av systemet som utløser en minnelekkasje og fryser hele applikasjonen hvis man prøver å submitte tekniske behov til band ved å trykke på "submit" knappen.

Arrangør På arrangørsiden kan du finne oversikten over sommer- og vinterfestivalen. Hvis du trykker deg inn på disse ser du oversikten over alle konserter og konsertinformasjon. Hvis du igjen trykker på en konsert får du opp en liste på høyre siden som viser hvilke teknikere som tilhører den konserten med kontaktinformasjon for teknikerne.

D Vedlegg - Testplan og Annet

D.1 Risikomatrise

Beskrivelse	Sannsynlighet (1-9)	Konsekvens(1-9)	Risiko	Forebyggende tiltak	Skadekontroll
Gruppemedlemmer kommer for sent til møte	9	3	27	Sette på vekkerklokke, innføre akademisk kvarter. Straff for å komme for sent, i form av kakestraff.	Etter tre forsentkomninger må vedkommende ta med kake og gjøre opp tapt arbeid.
Et gruppemedlem kan ikke komme pga. sykdom	5	3	15	Ha på seg nok klær nå det er kaldt, spise sunt og sove skikkelig.	Spørre om vedkommende kan jobbe hjemme eller fordele arbeidsoppgavene på de andre gruppemedlemmene.
Et gruppemedlem møter ikke opp gjentatte ganger over tid	4	7	28	Hvis man vet at man skal være borte over tid må man melde fra til gruppen.	Prøve å ordne slik at vedkommende kan jobbe med prosjektet selv om han/hun er borte. Fordele oppgavene på andre medlemmer.
Tap av kode eller annen data	2	8	16	Huske å ta regemessige backups av data.	Prøve å finne igjen kode som er så oppdatert som mulig å jobbe derfra. Bli enda flinkere til å ta backups-
Et gruppemedlem gjør av ukjent grunn ikke arbeidsoppgavene sine	3	7	21	Sørge for at alle er oppdatert på hva vi jobber med og hjelpe hverandre der det trengs.	Spør om vedkommende trenger hjelp til oppgaven sin og ha en god linje for kommunikasjon.
Vi møter på et uforutsett problem som stanser arbeidet	2	7	14	Tenke på hva som kan gå galt i fremtiden og være forberedt med løsninger.	Her må vi ta problemet som det kommer og jobbe sammen for å finne en løsning.

D.2 Testplan

Testplan - gruppe 18

For testing har teamet valgt å gjennomføre unit tester, integrasjonstester og system tester gjennom hele prosessen. Dersom dette lar seg gjennomføre med den begrensede tilgangen teamet har til kunden ønsker teamet også å gjennomføre en endelig akseptansetest med kunden etter eller i løpet av den siste sprinten for å kartlegge i hvilken grad produktet oppfyller kundens krav, både funksjonelle og ikke funksjonelle.
Tester utføres manuelt av teammedlemmer. De forskjellige test stadiene utføres av forskjellige teammedlemmer.

Unit test:

Her testes enkelt komponenter i systemet i isolasjon for å forsøke å oppdage feil i komponenten så tidlig som mulig og oppdage feil i komponenten før den integreres med resten av systemet.
Unit tester i prosjektet gjennomføres ved:

1. En komponent ved systemet fullføres
2. Et av teammedlemmene tester komponenten og forsøker å finne en tilstand der komponenten ikke oppfører seg slik den skal.
3. Dersom teammedlemmet ikke klarer å finne en feil i komponenten anses unit testen som bestått.

Integrasjonstest:

Etter unit testene utfører teamet integrasjons tester. Her testes større grupper med komponenter sammen.
Integrasjonstester i prosjektet gjennomføres ved:

1. Flere komponenter består unit testing og kombineres til en større modul.
2. Et av teammedlemmene tester den nye modulen og forsøker å finne tilstander der modulen oppfører seg annerledes enn forventet
3. Dersom ingen slike tilstander oppdages anses integrasjonstesten som bestått.

System test:

Ved jevne intervaller gjennomføres en total system test på alle deler av produktet som har bestått båre unit- og integrasjonstesting. Her er hensikten å teste at produktet oppfyller de kravene som er satt. Testene på dette nivået forsøker derfor å modellere systemet på et høyere nivå enn integrasjons og unit tester, for eksempel ved å teste funksjonelle og ikke funksjonelle krav til systemet.