Bacheloroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

Erik Borgeteien Hansen
Oda Alida Fønstelien Hjelljord

Utvikling av brukervennlig mellomvare

Unity integrasjon med SensMax TAC-B mennesketellende radarsensor

Bacheloroppgave i Dataingeniør
Veileder: Elise Klæbo Vonstad
Mai 2023

NTNU
Kunnskap for en bedre verden

Erik Borgeteien Hansen
Oda Alida Fønstelien Hjelljord

Utvikling av brukervennlig mellomvare

Unity integrasjon med SensMax TAC-B mennesketellende radarsensor

Bacheloroppgave i Dataingeniør
Veileder: Elise Klæbo Vonstad
Mai 2023

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU

Kunnskap for en bedre verden

Sammendrag

Oppgaven undersøkte hvordan man kan implementere en programvarepakke til den etablerte spillmotoren Unity på en brukervennlig måte. Hensikten med programvarepakken var å tilrettelegge for og forenkle bruken av SensMax TAC-B mennesketellende radarsensor i spill og kunstprosjekter. Oppdragsgiver, Able Magic AS, håpet med dette å kunne korte ned på utviklingstiden på sine prosjekter hvor de tar i bruk sensoren.

Utviklere har et intuitivt forhold til begrepet "brukervennlighet", men i denne oppgaven har det blitt utforsket hva det vil si å utvikle brukervennlig programvare for andre utviklere. Det har blitt sett på to ulike rammeverk for å bedømme brukervennlighet; brukertesting og kognitive dimensjoner. En sammenligning mellom hva teorien sier om brukervennlighet, og hvordan utviklere tenker på brukervennlighet i praksis har blitt gjort.

Det har blitt vanlig praksis er å etterlate brukervennlighetsanalyser til etter utviklingsfasen, men studier som ble gjennomgått i oppgaven indikerer at fokus på brukervennlighet gjennom hele utviklingsprosessen reduserer brukervennlighetsproblemer i sluttproduktet.

Teorien rundt brukervennlig mellomvare ga et utgangspunkt i prosjektet for å utvikle pakken. Fokuset var på å utvikle en pakke som er brukervennlig å bruke og mulig for andre å videreutvikle. Pakken er grundig testet gjennom brukertesting, og har iterativt blitt forbedret etter testene.

Resultatet av oppgaven er en programvarepakke som tilbyr all den grunnleggende funksjonaliteten til sensoren. Denne pakken er brukervennlig og vil være et godt eksempel for andre som ønsker å utvikle lignende produkt.

Abstract

This thesis investigated how to implement a software package for the established game engine Unity in a user-friendly manner. The purpose of the software package was to facilitate and simplify the use of the SensMax TAC-B human counting radar sensor in game and art projects. The client, Able Magic AS, hoped that this would shorten the development time for their projects that utilize the sensor.

Developers have an intuitive understanding of the term "user-friendly", but in this text we have explored what it means to develop in a user-friendly towards other developers. Two different frameworks have been examined to assess user-friendliness: user testing and cognitive dimensions. A comparison has been made between what theory says about user-friendliness and how developers perceive user-friendliness in practice.

It has become common practice to leave usability analysis until after the development phase, but studies reviewed in this study indicate that focusing on usability throughout the development process reduces usability issues in the final product.

The theory of user-friendly middleware provided a starting point in the project for developing the package. The focus was on creating a package that is user-friendly to use and possible for others to develop further. The package has been thoroughly tested through user testing and has been iteratively improved upon based on the tests.

The result of the study is a software package that offers all the basic functionality of the sensor. This package is user-friendly and will serve as a good example for others who wish to develop similar products.

Forord

Oppgaven vekket interessen vår da den tilbød en unik mulighet til å utvikle opp i mot maskinvare. Vi har fått muligheten til å utforske to vidt forskjellige nivåer av utvikling; vi har arbeidet med nettverksprotokoller, og vi har jobbet mer abstrakt med den veletablerte spillmotoren Unity. Vi har lest grunnleggende RFC-dokumenter for å forstå hvordan MQTT-protokollen virker, og vi har sett videoer som forklarer hvordan forhåndskompileringen til Unity virker. Alt i alt har prosjektet vært svært variert, til tider krevende, men det har samtidig vært svært givende. Vi sitter igjen med en unik kompetanse.

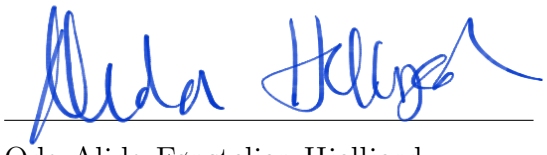
Gjennom prosessen har det blitt utviklet en brukervennlig programvarepakke til Unity, som lar deg kommunisere med SensMax TAC-B sensorer i Unity-prosjekter via MQTT-protokollen over wifi. Denne ble utviklet i sprinter med brukertester for å lage et best mulig produkt. Takk til Fredrik Eiding, Felix Albrigtsen, Tore Bergebakken, Petter Rosvoll, Aleksander Halvorsen Holthe, Peder Jørgen Nagelsaker Lexau, Rinsai Moontika og alle andre som stilte opp på brukertester. Produktet vårt hadde ikke vært det samme uten.

Vi var ikke kjent med Ablemagic før vi begynte arbeidet på oppgaven, men under arbeidet har vi blitt kjent med en veldig hyggelig gjeng. Ablemagics folk har stilt opp på brukertester og veiledet arbeidet. Takk til Ablemagic for kontorplass. Takk spesielt til Hallgeir Løkken for oppfølging, hjelp med Unity og tilbakemelding på koden.

Takk til Elise Klæbo Vonstad for hjelp med skriving av denne oppgaven, oppfølging og svar på alle mulige spørsmål. Spesielt takk for hjelpen med å utvikle problemstillingen i denne oppgaven.

Takk til Sondre Rokstad Grimsmo for korekturlesing av denne oppgaven.

Erik Borgeteien Hansen



Oda Alida Fønstelien Hjelljord

Trondheim, 22. mai 2023

Oppgavetekst

Oppgaven som ble utlyst til gruppen var å implementere en pakke i Unity som fasiliterer kommunikasjon mellom Unity og SensMax TAC-B mennesketellende radarsensor (sensoren).

Etter oppstartsmøte skrev Hallgeir Løkken en utvidet kravspesifikasjon på vegne av Ablemagic, vedlegg A. Denne ble utgangspunktet til systemutviklingsdelen av denne oppgaven, men gruppen sto fritt og var oppfordret til å utvide på denne. Ablemagic etterspurte blant annet en gjennomgang av MQTT-protokollen til sensoren og en visualisering av objektene sensoren detekterer i Unity. I kravspesifikasjonen uttrykte Ablemagic en ambisjon om å få testet sensorens evner i vanskeligere situasjoner [A]. Dette ønsket ble nedprioritet da det naturlig ligger utenfor domenet til oppgaven og systemutviklingen gruppen fokuserte på. Funn om sensorens grenseverdier gjort underveis i prosjektet ble rapportert til Ablemagic, men vil ikke være del omfanget til oppgaven.

Siden gruppen sto relativt fritt til å utvide programvaren fra oppdragsgiver sin side ble det bestemt en rekke andre krav til systemet beskrevet i visjonsdokumentet for prosjektet, vedlegg C. Krav for programvaren fremlagt i Ablemagics kravspesifikasjon ble sett på som enkleste brukbare produkt (MVP).

Innhold

Figurliste

Tabelliste

1 Introduksjon og relevans

1.1 Bakgrunn

Ablemagic ønsket en utvidelse til Unity som kunne håndtere kommunikasjon mellom spillmotoren Unity og Sensmax TAC-B 3D-WP mennesketellende radarsensor. Dette er for å bruke i sine interaktive installasjoner. De lager interaktive installasjoner både som kunst og for underholdning. Noen av disse skal bare være aktive når det er mennesker i nærheten, og da er en radarsensor en enkel og driftsikker måte å oppdage mennesker på. Slike installasjoner blir ofte laget i Unity, og en pakke som er enkel i bruk vil dermed effektivisere utviklingsprosessen deres.

Andre spillutviklere kan også være interessert i denne pakken dersom de skal benytte sensoren i sine prosjekter. Det finnes pakker som håndterer MQTT til Unity, men disse koster penger og funksjonalitet for å benytte sensoren må fortsatt legges til disse. Det er et behov for en pakke med åpen kildekode som håndterer MQTT og er spesielt tilpasset sensoren. Derfor skal denne pakken tilbys som åpen kildekode slik at alle utviklere som ønsker å ta i bruk pakken kan gjøre dette gratis.

Brukervennlighet er viktig i mellomvare, da det vil effektivisere utviklingen og øke sjansen for at andre tar i bruke programmet. En godt designet og dokumentert Unity pakke vil være et godt utgangspunkt for videreutvikling. Alle som programmerer har opplevd frustrasjonen med å jobbe med et programvarebibliotek som er dårlig dokumentert og vanskelig å sette seg inn. Gruppen ønsket å lage en pakke som var brukervennlig, effektiv og enkel for andre å videreutvikle.

Et mål ved dette prosjektet er å utvikle en programvarepakke som kan fungere som et godt utgangspunkt og en inspirasjonskilde til utviklere som vil lage lignende pakker, for eksempel pakker som bruker en lignende sensor eller andre IoT-enheter som benytter seg av MQTT-protokollen.

1.2 Problemstilling

Siden gruppen sto relativt fritt i videre utvikling av systemet [A] jobbet vi med veileder Elise Klæbo Vonstad for å utvikle en problemstilling som fokus for oppgaven. Problemstillingen som ble kommet frem til var:

"Hvordan kan man utvikle programvare som er brukbar for andre utviklere?"

Denne problemstillingen vil være sentral i utviklingen og forskningen i oppgaven, og vil bli utforsket gjennom litteraturstudier og brukertesting. Mye av fokuset til denne teksten vil dermed være på å utforske hva det er som gjør et programvarebibliotek brukbart og brukervennlig.

1.3 Dokumentstruktur

Kapittel 1 - Introduksjon og relevans beskriver bakgrunnen og problemstillingen i oppgaven, og gir kontekst for det videre fokuset i teksten.

Kapittel 2 - Teori og relevant litteratur fremlegger det teoretiske grunnlaget i oppgaven.

Kapittel 3 - Metode tar for seg hvordan gruppen har jobbet med oppgaven. Det legges frem arbeidsmetode, teknologivalg og vitenskapeligmetode.

Kapittel 4 - Resultater presenterer hva gruppen har oppnådd i relasjon til målene satt tidlig i prosjektet.

Kapittel 5 - Diskusjon drøfter om problemstillingen er besvart og hvorfor resultatene ble som de ble.

Kapittel 6 - Konklusjon og videre arbeid beskriver konklusjonene som kan trekkes fra arbeidet og videre arbeid som kan gjøres med produktet og i feltet som helhet.

1.3.1 Benevnelser

Tabell 1: Forklaring av nomenklatur brukt i dette dokumentet

Forkortelse	Forklaring
MVP	Minimum viable product (enkleste brukbare produkt)
Sensoren	SensMax TAC-B 3D-WP mennesketellende radarsensor [3].

Forkortelse	Forklaring
KD	Cognitive Dimensions of Notations (Kognitive Dimensjoner av Notasjoner) [4]
MQTT	Sto opprinnelig for Message Queuing Telemetry Transport, men betydningen av akronymet er blitt gått vekk fra i senere tid. MQTT er en kommunikasjonsprotokoll som brukes i tingenes internett [5].
JSON	JavaScript Object Notation, en måte å effektivt sende objekter som tekst [6].
IoT	Internet of things (tingenes internett), en felles betegnelse for kommunikasjon over nett mellom elektroniske enheter [7].
SSH	Secure Shell, en protokoll som gir sikker tilgang til en fjern datamaskin over et usikret nettverk [8].
HTTP	HyperText Transfer Protocol, en familie med tilstandsløse forespørsel/svar-protokoller som tillater kommunikasjon med hypertekst-informasjonssystemer [9].
HTTPS	HyperText Transfer Protocol Secure, en utvidelse av HTTP som benytter kryptering for å sikre kommunikasjon [10].
Prefab	Prefabrikkert, refererer i dette dokumentet til ferdiglage spillobjekter i Unity.
API	Application programming interface (programmeringsgrensesnitt) er et grensesnitt i en programvare som gjør at spesifikke deler av denne kan aktiveres fra en annen programvare [11].
CICD	Continual integration and continual delivery/deployment (kontinuerlig integrasjon og kontinuerlig leveranse/utrulling) er en praksis som automatiserer enkelte deler av utviklingsprosessen for å kunne levere kode mer effektivt.

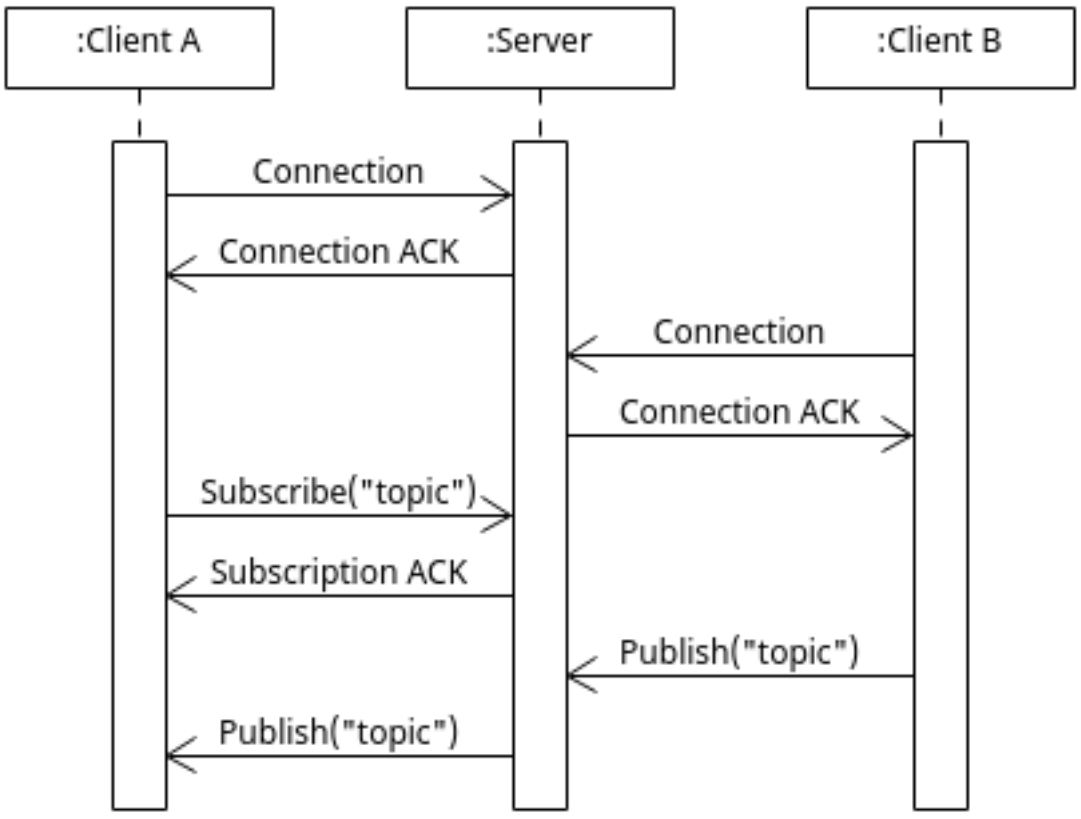
Forkortelse	Forklaring
WCAG	Web Content Accessibility Guidelines er et sett av rettningslinjer for universell utforming på nettsider [12].
README	En "Read me"-fil er en fil gjerne i Markdown format som beskriver et kode-prosjekt. Den inneholder gjerne installasjonsguide og forklaring av enkel bruk.

2 Teori og relevant litteratur

2.1 MQTT

MQTT er en protokoll for meldingstransport på applikasjonslaget, i likhet med HTTP. Den beskriver en klient/server-arkitektur som benytter publisering og abonnering (publiser/abonner) for å kommunisere. Protokollen ble designet for å være lettvektig, åpen, og enkel å implementere. Dette gjør den til en idéell kandidat i situasjoner med begrensninger, for eksempel i maskin-til-maskin kommunikasjon eller tingenes internett (IoT) der båndbredde og forsinkelse kan være til hinder. [13]

2.1.1 Publiser/abonner-modellen



Figur 1: Sekvensdiagram som viser hvordan publiser/abonner-modellen virker

I motsetning til den mer tradisjonelle klient/server-modellen hvor hver klient tar direkte kontakt med endepunktene, benytter publiser/abonner-modellen en sentral server som

legger opp til indirekte kommunikasjon mellom de oppkoblede klientene. Hver enkelt klient vet bare om serveren, og kan enten publisere eller abonnere på meldinger til eller fra serveren [14].

2.1.2 Skalerbarhet

I situasjoner der man kan gjenbruke koblingen mellom en klient og server over mange pakker, vil publiser/abonner-modellen skalere bedre med hensyn på responstid og mengde data per oppdatering enn forespørsel/svar-modellen til HTTP. Dette er på grunn av at HTTP er nødt til å sende metadata om tilkoblingen med hver eneste melding, mens MQTT-protokollen etablerer dette på starten av tilkoblingen [15].

2.1.3 Meldingsfiltrering

Serveren organiserer meldingene som passerer via emner. Et emne er en hierarkisk struktur, i likhet med et filsystem, og blir representert som en tekstreng med skråstrek som skilletegn mellom ordene. Hver melding i denne protokollen har et emne. Klienter som er koblet til serveren kan abonnere på emner, og hver gang en melding som har riktig emne ankommer serveren, sender serveren meldingen videre til de abonnerte klientene [14].

2.2 Programvarebibliotek

Et programvarebibliotek er en samling av ressurser man kan gjenbruke når man programmerer. Innholdet kan være funksjoner, klasser, data, konfigurasjoner, skript og lignende. Biblioteker brukes primært til å gjenbruke komponenter i stedet for å skrive kode fra bunnen av. Programvarebiblioteker er en god ressurs for programmere; det er mulig å studere bibliotekkomponenter for å bli bedre kjent med et språk, en programmeringsstil eller bruksmønstre. Et av de største vanskelighetene med å bruke programvarebiblioteker, spesielt når bibliotekene blir større, er tilgjengeligheten av gode verktøy for organisering, navigering og gjenfinning i biblioteker [16].

Mange programvarebiblioteker har blitt testet, oppdatert og vedlikeholdt lenge. Dette har bidratt til å eliminere mange av feilene som egne løsninger kan ha før disse blir oppdaget. Det kan derfor lønne seg å benytte et vellaget programvarebibliotek over å implementere den samme funksjonaliteten på nytt [16].

2.3 API

Et programmeringsgrensesnitt (API) kan sees på som en bro mellom to systemer. Ved å benytte de kan du utnytte data og metoder fra et annet system uten å vite noe om implementasjonen dens. API-er blir ofte brukt av programvare for å kommunisere med annen programvare, og ikke sluttbrukere. For å bruke et API sender man en forspørsel i et format avtalt på forhånd, gjerne i dokumentasjonen, og så får du svar fra API-et som du kan benytte [17].

API-er er lignende til programvarebiblioteker i at de tillater utviklere å benytte kode skrevet av andre direkte i kode. Forskjellen mellom disse er imidlertid at når et programvarebibliotek må importeres i prosjektet vil API-ers metoder kalles eksternt og kjøres på API-ets vertsmaskin [17].

2.4 Spillmotorer

En spillmotor er et programvarerammeverk fortrinnsvis laget for å utvikle spill. En spillmotor inneholder relevante biblioteker og støtteprogrammer for spillutvikling. Dette kan for eksempel være biblioteker for 2D og 3D grafikk, fysikksimulering, lydavspilling, brukergrensesnitt, animasjoner eller annet [18]. Det finnes mange forskjellige spillmotorer som bruker flere forskjellige programspåk [19].

2.5 Serialisering

JavaScript Object Notation (JSON) er en metode for dataoverføring som er leselig av både maskiner og mennesker. Selvom navnet inneholder JavaScript er JSON uavhengig av programmeringsspråk og er en vanlig data overførings metode som er brukt i en rekke applikasjoner. JSON er innlemmet i nesten alle web-tjenester som en enkel og effektiv måte å overføre data mellom frontend og backend. JSON representerer data på to måter: Objekt og Array. Dette gjør det mulig å overføre objekter og lister fra et programmeringsspråk til et annet, noe som gjør JSON til et naturlig valg i dataoverføring og lagring [6].

2.6 Kontinuerlig integrasjon og leveranse

Kontinuerlig integrasjon og leveranse er en utbredt programmeringspraksis som har som formål å automatisere deler av utrullingsprosessen. Dette korter ned på tiden utviklingssyklusen tar, og lar utviklere levere bedre kode raskere [20].

Kontinuerlig integrasjon er en metode hvor utviklere integrerer og fletter koden sin inn i hovedgrenen ofte. Dette innebærer ofte også automatisk bygging og testing av koden, og gir gjerne utviklerne rask tilbakemelding om noe går galt.

Det neste skrittet i prosessen er kontinuerlig leveranse. Kontinuerlig leveranse sørger for alltid å ha produksjonsklar kode tilgjengelig, som i teorien kan rulles ut på serverene. Dette sikrer lavere kostnader, raskere tilbakemelding fra bruker, og lavere risiko når man skal rulle ut koden sin.

Et annet alternativ til kontinuerlig leveranse er kontinuerlig utrulling, som tar hele prosessen et skritt videre og automatisk ruller ut koden dersom alt av tester og andre kriterier passerer [21].

2.7 Åpen kildekode

Åpen kildekode er kildekode som er gjort fritt tilgjengelig for mulig endring og redistribusjon. Selv om åpen kildekode referer til kildekode i navnet et det tenkt til å gjelde alle produkter i disse til fellen er det ment at teknologiens design skal være tilgjengelig for reproduksjon. Produkter som kan være åpen kildekode inkluderer tillatelse til å bruke kildekoden, design-dokumenter eller innholdet i produktet. Det åpne kildekode-modellen er en desentralisert programvareutviklingsmodell som oppmuntrer til åpent samarbeid. Et hovedprinsipp for utvikling av åpen kildekode er likeverdig produksjon, der produkter som kildekode, blåkopier og dokumentasjon er fritt tilgjengelig for allmennheten. Bevegelsen for åpen kildekode i programvare startet som en reaksjon på begrensningene ved proprietær kode [22].

Open Source Initiative [23] legger frem at ikke bare tilgang til kildekode er nødvendig for at et produkt skal være åpen kildekode. De legger frem 10 krav for at et produkt skal kunne kalles åpen kildekode [23]:

1. Fri redistribusjon

Åpen kildekode legger ingen føringer på hvordan koden kan redistribueres og tar ingen avgift på salg av produkter der koden er brukt.

2. Kildekode
   Kildekoden må være tillgjengelig for alle interesenter, enten distribuert sammen med det kompilerte programmet eller lett tilgjengelig gjennom andre kilder.

3. Modifiserte verk
   Lisensen må tillate modifikasjoner og avledede verk og ikke legge føringer på hvordan disse verkene kan distribueres.

4. Integritet til opphavspersonens kildekode
   Lisensen kan begrense distribusjonen av kildekoden i endret form bare hvis lisensen tillater distribusjon av "patch-filer" sammen med kildekoden, slik at programmet kan modifiseres ved byggetid.

5. Ingen diskriminering mot enkeltpersoner eller grupper
   Lisensen skal ikke diskriminere mot noen enkeltperson eller gruppe av personer.

6. Ingen diskriminering mot virksomhetsområder
   Lisensen skal ikke begrense noen fra å bruke programmet innen et bestemt virksomhetsområde.

7. Distribusjon av lisens
   Rettighetene knyttet til programmet må gjelde for alle som programmet redistribueres til, uten behov for utstedelse av en ekstra lisens fra disse partene.

8. Lisensen må ikke være spesifikk for et produkt
   Rettighetene knyttet til programmet må ikke avhenge av at programmet er en del av en bestemt programvaredistribusjon.

9. Lisensen må ikke begrense annen programvare
   Lisensen må ikke legge begrensninger på annen programvare som distribueres sammen med den lisensierte programvaren.

10. Lisensen må være teknologinøytral
    Ingen bestemmelse i lisensen kan baseres på en bestemt teknologi eller grensesnittstil.

2.8 Ren kode

Ren kode er en programmeringspraksis som fremmer produksjonen av kode som er lett å lese, forstå og vedlikeholde. Dette innebærer at koden skal være funksjonell, veldokumentert, og følge klare standarder og prinsipper for kodekvalitet.

Det finnes ulike teknikker og prinsipper som kan brukes for å skrive ren og lesbar kode. Dette inkluderer men er ikke begrenset til; begrense funksjoner til å bare utføre én oppgave, minimere bruken av kommentarer ved å skrive tydelig kode som er enkel å forstå, ha selvforklarende navngivning av variabler og funksjoner. [24].

Det vil være spesielt nytting å strebe etter ren kode i prosjekter der videreutvikling skal gjøres av andre utviklere enn de som originalt lagde produktet. Det er derfor spesielt nyttig i åpen kildekode [25].

2.8.1 Tredjepartskode

I boken "Clean Code" [24], beskriver Martin gnisningen som oppstår i bruken av programmvarebibliotek. Når utviklere lager et programvarebibliotek etterstreves bred anvendelighet slik at programmet kan fungere i mange miljøer og appellere til et bredt publikum. En bruker anvender biblioteket til en spesifikk oppgave og ønsker dermed at den skal være fokusert på deres spesielle behov. Ren tredjepartskode minimerer denne gnisningen.

2.9 Brukervennlighet

Brukervennlighet kan beskrives som hvor lett en person kan bruke et produkt under konkrete omstendigheter [1]. Dette innebærer å ta hensyn til brukerens evner, erfaring, og målsettinger, samt konteksten der produktet blir brukt [26]. Brukervennlighet er mer formelt definert av International Organization for Standardization som "i hvilken grad et produkt kan brukes av spesifiserte brukere for å oppnå spesifiserte mål med hastighet, effektivitet og tilfredshet i en spesifisert brukssammenheng" [27]. Brukervennlighet defineres vanligvis med fem hovedkonsepter [1]:

1. Lærbarhet. Hvor lett er produktet å lære seg å bruke for første gang?

2. Effektivitet. Etter at en bruker har blitt kjent med produktet, hvor effektivt kan de utføre ulike funksjoner?

3. Memorerbarhet. Hvis en bruker ikke bruker programmet på lenge, hvor lett er det for dem å huske hvordan det brukes?

4. Feil. Hvor ofte støter brukeren på feil i programmet?

5. Fornøydhet. Hvor fornøyd er brukeren med produktet?

I boken "Usability Testing Essentials" fremhever Barnum betydningen av brukere, mål og kontekst ved vurdering av brukervennlighet [28]. Viktigheten av hver av disse blir forklart med eksempler under.

Brukere: Hvem som er brukerne av et program påvirker brukervennligheten [26]. Hva som er brukervennlig for et barn er ikke nødvendigvis det som er brukervennlig for en voksen. Et barn som enda ikke er stødig i lesing vil foretrekke bilder som indikasjon over tekst, mens en voksen vil kanskje synes tekst er mer presist og utvetydig.

Mål: Hva et program prøver å oppnå påvirker hvilke funksjoner som øker eller senker brukevennlighet [26]. En utdanningsressurs designet for å hjelpe en student å få oversikt i et fag tidlig i semesteret, som inneholder detaljert informasjon, men ikke gir oppsummeringssider, kan være svært informativ, men til slutt ubrukelig som en tidlig ressurs. Når en student prøver å gjøre seg klar for forelesninger har de ikke nytte av alle detaljene, men heller en generell oversikt. Det blir dermed mindre sannsynlig at de benytter ressursen.

Kontekst: Miljøet eler konteksten et program skal operere i påvirker brukervennligheten [26]. Hvis et program kjører på en maskin med langt mindre resurser enn programmet er designet for vil det operere med forsinkelser og oppleves mindre brukervennlig.

2.9.1 Viktigheten av brukervennlighet

Det er vist at brukervennlighet er en av de største faktorene som tilsier om en bruker aksepterer et nytt digitalt verktøy [29], som predikerer faktisk bruk. Brukervennlighet er derfor viktig for alle digitale ressurser.

Det er ofte en kommersiell baktanke når brukervennlighet er i fokus, for eksempel for en butikkside blir det begrunnet som "hvis en bruker ikke finner et produkt kan de ikke

kjøpe det" [30]. I avsnittet under vil pedagogiske ressurser trekkes frem som et eksempel på dette.

Brukervennlighet er relevant i utformingen av læringsplatformer der brukerne blir nødt til å samhandle med komplekse systemer som de ikke allerede er kjent med. Den vanskeligste delen av et kurs bør alltid være innholdet og ikke hvordan stoffet blir presentert. I næringslivet har effektiviteten til nettkurs blitt grundig studert. Bedrifter erstatter i stor grad tradisjonelle kurs med nettkurs som er billige og praktiske. Nettkurs har imidlertid høyere frafall enn tradisjonelle kurs. Én grunn til dette kan være dårlig brukervennlighet i ressursene [26].

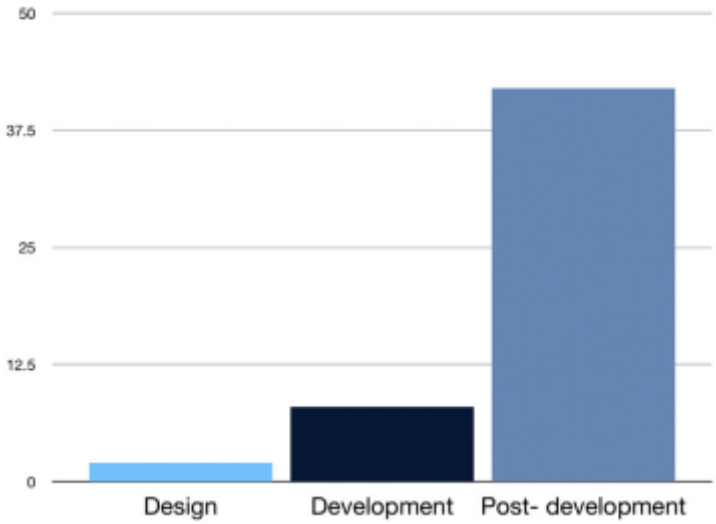
2.9.2 Brukervennlighet i litteratur og praksis

I metastudiet om brukervennlighetsanalyse av Rauf et al. [1] ble det gjennomgått en rekke brukervennlighetsanalyser. Disse viste at selv om de fem konseptene, beskrevet i seksjon 2.9, er mest vanlig er det mange andre konsepter som kan innlemmes i analyser. I studiet ble det gjort en gjennomgang av hvilke prinsipper som ble fokusert på. Fokuset var på nyansene mellom brukervennlighet i litteraturen og i brukerundersøkelser i praksis. I figur 2 vises en sammenligning av ordbruk i litteratur og praksis [1]. Dette viser at det er en forskjell mellom hva litteraturen beskriver om brukervennlighet og hva utviklere fokuserer på når de analyserer brukervennlighet.

(a) Mest brukte termer i litteratur (b) Mest brukte termer i analyser i felten

Figur 2: Sammenligning av ordbruk i litteratur og i felten [1]

Det ble funnet i studien at det er vanligst å gjøre brukervennlighetsanalyser sent i, eller etter utviklingsprosjektet er ferdig [1]. Bhaskar et al. [31] viste imidlertid at et gjennomgående fokus på brukervennlighet på alle stadiene av utvikling kunne resultere i programvare med få brukervennlighetproblemer før første brukertester er gjennomført.



Figur 3: Graf over når brukervennlighetsanalyser utføres [1]

2.9.3 Universell utforming

Universell utforming innebærer å produsere produkter, omgivelser, dokumenter, programmer og tjenester slik at de kan brukes av så mange mennesker som mulig, uavhengig av funksjonsevne. Universell uforming er viktig i alle produkter. Web Content Accessibility Guidelines (WCAG) fremlegger rettningslinjer for Universell uforming av nettsider. Disse rettningslinjene er et godt utgangspunt for vudrering av uformingen av alle digitale brukergrensesnitt. WCAG består av fire hovedprinsipper [12]:

Mulig å oppfatte. Informasjon skal kunne presenteres på en måte brukerne kan oppfatte. Dette betyr at informasjonen må være tilgjengelig på en måte som kan oppfattes av to eller flere sanser. Eksempler på tilrettelegning for dette er å ha køy kontrast mellom tekst å bakgrunn og alternativ tekst til bilder.

Mulig å betjene. Brukere skal kunne interagere programmet med det utstyret de benytter. Dette betyr at brukere som for eksempel er avhengig av tastatur for å interagere med elementer i grensesnittet. Dette betyr at all funksjonalitet er tilgjengelig med bare tastatur eller bare mus.

Forståelig Prinsippet handler om presist, enkelt språk og god hjelpefunksjonalitet. Kriterier som kan knyttes til dette er samsvar mellom koding og sidespråk, slik at teksten blir lest opp på rett måte for dem som bruker talesyntese.

Robust Omhandler at koden må være utformet slik at den kan brukes når man introduserer annen teknologi, for eksempel kompenserende teknologi som skjermlesere. Dette blir oftest ivaretatt ved bruk av standardelementer.

2.9.4 Kognitive dimensjoner

En annen modell for brukervennlighet er Kognitive Dimensjoner av Notasjoner Rammeverket (KD).

KD er et rammeverk for å beskrive brukervennligheten til notasjonssystemer som programmeringsspråk, tegneprogrammer og musikknotasjon. Det kan også brukes til å beskrive informasjonsartefakter som klokker, radioer og termostater [4]. Systemet er et forsøk på å utvikle et universelt system for brukervennlighet som er uavhengig av brukergrensenitt og kontekst. I en av de grunnleggende tekstene om brukervennlighet i programmeringsspråk [32] bennyttes KD til å sammenligne tekstbaserte programmeringsspråk med visuelle programmeringsspråk.

I motsetting til mange andre rammeverk fokuserer KD på prosessen mer enn det ferdige produktet [1]. Dette systemet benyttes gjerne under utviklingsprosessen og suksessen evalueres i etterkant ved brukertesting eller andre evaluering metoder.

Målet med KD-rammeverket er å gi et vokabular til designere når de undersøker brukervennlighetskonsekvensene av designbeslutningene sine. Designere av notasjonssystemer er klar over at beslutningene deres har en innvirkning på brukervennligheten til systemet, men mange designere kjenner bare til dette på en intuitiv måte [4]. Det er da gunstig å ha et system for å diskutere brukervennlighet i notasjonssystemer. I tillegg til å være et verktøy for diskusjon av brukervennlighet er KD også et rammeverk for å tenke på naturen til notasjonssystemer og hvordan folk interagerer med dem.

I KD defineres 14 dimensjoner. Mange av disse er motstridende hverandre, det vil si at å forbedre en vil forverre en annen. Dette innebærer at utviklere må balansere forskjellige dimensjoner for å bygge et brukervennlig system som møter deres behov.

Viskositet: Motstand mot endring. Et viskøst system krever mange brukerhandlinger

for å oppnå ett mål. For eksempel å endre alle overskrifter i en tekst til store bokstaver kan kreve en handling per overskrift [4].

Synlighet: Evnen til å se komponenter lett. Et system med høy synlighet er lett å navigere, men tillater få abstraksjoner da disse skjuler komponenter [4].

Tidlig binding: Begrensninger på rekkefølgen av handlinger. Hvis systemet tvinger brukeren til å velge verktøy før de forstår oppgaven, kan brukeren velge feil og måtte starte på nytt fra starten [4].

Skjulte avhengigheter: Viktige koblinger mellom enheter er ikke synlige. Hvis en enhet henviser til en annen enhet, som igjen henviser til en tredje, kan endring av verdien til den tredje enheten ha uventede konsekvenser [4].

Rolle-uttrykksevne: Formålet med komponenter er tydelig. Rolle-ekspressive notasjoner gjør det enkelt å oppdage hvorfor programmereren har bygget strukturen på en bestemt måte [4].

Feilfølsomhet: Feilfølsomme notasjoner støter ofte på feil og/eller feil som oppstår har store negative konsekvenser [4].

Abstraksjon: Typer og tilgjengelighet av abstraksjonsmekanismer. Abstraksjoner, eller omdefineringer, endrer den underliggende notasjonen. For eksempel makroer, datatyper og globale søk-og-erstatt-kommandoer [4].

Sekundær notasjon: Ekstra informasjon som ikke er del av notasjonenes formelle syntaks. I stedet for å prøve å forutse hver eneste mulige brukerbehov, kan systemer støtte sekundære notater. Programmeringsspråk tilbyr ofter sekundær notasjoner som kommentarer og dokumentasjonsstrenger [4].

Nærhet til oppgave: Hvor nær representasjonen av grensesnittet er til domenet de representerer. Et eksempel på dårlig nærhet til oppgave er dersom en knapp sier "legg i handlekurv", men å trykke på den fører til at elementet blir lagt i ønskelisten [4].

Konsistens: Lik betydning er uttrykt på en liknende syntaktisk form. Det er dette prinsippet gettere og settere kommer fra for eksempel. Det er ingenting som stopper en utvikler fra å kalle getterene sine noe annet enn "getVariable", annet enn at det blir konsistenst og dermed mer brukervennlig å ha samme navn på samme ting [4].

Diffushet: Språklig oppblåsthet. Noen notasjoner kan være irriterende langtekkelige

eller oppta for mye verdifull plass på en skjerm. Store ikoner og lange ord reduserer det tilgjengelige arbeidsområdet [4].

Vanskelige mentale operasjoner: Høyt krav til kognitive ressurser. En notasjon kan gjøre ting komplekse eller vanskelige å regne ut i hodet, ved å kreve overdreven belastning på arbeidsminnet til brukere [4].

Provisjonalitet: Grad av forpliktelse til handlinger eller markeringer. Dette referer til hvor enkelt det er å ta tilbake en handling. For eksempel å skrive med blyant istedet for penn for å kunne viske vekk feil. Et tekstredigeringsprogram som ikke har en angre funksjon ville foreksempelt ikke bli sett på som veldig brukervennlig [4].

Progressiv vurdering: Arbeidet hittil kan sjekkes når som helst. Evaluering er en viktig del av designprosessen, og notasjonssystemer kan lette evaluering ved å tillate brukere å stoppe midtveis for å sjekke arbeidet så langt [4]. Muligheten til å kompilere og kjøre uferdig kode er et eksempel på progressiv vurdering.

2.10 Brukertesting

I løpet av utviklingsprosessen er det enkelt å anta at produktet som blir utviklet er brukervennlig siden man som utvikler er kjent med hvordan alle funksjonene er satt opp. Barnum skriver i sin bok om temaet: "Fra det øyeblikket du vet nok til å snakke om et produkt [...] vet du for mye til å kunne avgjøre om produktet ville være brukervennlig for en person som ikke vet det du vet" [28]. Det er derfor ikke bare nødvending å vurdere brukervennlighet gjennom utviklingsprossesen men også teste den på brukere som ikke har erfaring med systemet fra før [26].

Brukertesting er en av de mest populære metodene å oppnå brukervennlighet brukt idag [1]. I "The Human-Computer Interaction Handbook" legger Sears og Jacko [33] frem at for å gi valide resultater må en brukertest oppnå seks kriterier:

1. Det er et produkt eller system å teste.
2. Fokuset til testen er på brukvennlighet.
3. Deltakerene i testen er sluttbrukere eller potensielle sluttbrukere.
4. Testdeltakeren gjør oppgaver og forklarer tankeprosessen sin til testeren.

5. Resultatene blir notert og analysert.

6. Resultatene formidles til rett gruppe.

2.10.1 Fokuset til testen

Det fremstår kanskje som en selvfølge at brukervennlighetstester skal fokusere på brukervennlighet, men det er ikke uvanlig å blande inn markedsføringsspørsmål i brukertester. Dette vil ikke bidra i å teste brukervennlighet og vil ikke gi nyttige svar om markedsførbarheten til produktet. Siden brukertester gjerne har få deltakere vil spørsmål som "Hadde du vært interesert i å kjøpe dette produktet?" ikke gi signifikante svar. Dette er blant annet fordi brukere som er villig til å møte opp for brukertesting, er mer interessert i produktet enn en gjennomsnittlig bruker. I dette tilfellet hadde det vert mer hensiktsmessig med for eksempel en spørreundersøkelse [33].

2.10.2 Testbrukere

For at testen skal gi generaliserbare resultater må testpersonene være sluttbrukere eller potensielle sluttbrukere. Det er da nyttig å lage en brukerprofil som beskriver egenskaper brukere har til felles og, egenskaper som er forskjellige blant brukere [33].

2.10.3 Oppgaver

I gjennomføringen av brukertester brukes oppgaver til å dirigere testpersonen. Dette bidrar med å teste om brukergrensesnittet er intuitivt og om funksjoner er robuste for brukerinndata. Det er sjeldent mulighet til å teste alle funksjonene til et system, derfor er det nødvendig å velge oppgaver for å best teste systemet[33]. Når man skal velge oppgaver bør man inkludere:

1. Oppgaver som tester funksjoner som brukes ofte eller er kritiske for systemet.

2. Oppgaver som tester funksjoner som har høy sannsynlighet for å støte på brukerproblemer

3. Oppgaver som tvinger brukeren til å navigere dypt i systemet. Målet er å inkludere oppgaver som øker grundigheten til testen.

Når man lager oppgavene er det nyttig å lage oppgavescenarier, dette hjelper testpersonen å forstå oppgavens intensjon bedre [33]. Et oppgavescenario kan for eksempel være:

"Du har nettopp kjøpt en ny telefon, boksen ligger på bordet. Ta produktet ut av boksen og sett det opp sånn at du kan motta telefonsamtaler."

2.10.4 Gjennomføring

Under gjennomføringen av brukertesten får testpersonen oppgaver som beskrevet over. Testpersonen forsøker å løse disse med programmet som skal testes. I løpet av denne prosessen vil testeren se funksjoner som ikke er intuitive og eventuelle feil i programmet.

Noen ganger er det nødvendig for testeren å hjelpe testpersonen med en oppgave slik at videre funksjonalitet kan bli testet. Om en meny ikke fungerer og slik hindrer testpersonen i å teste funksjonen som menyen skulle lede til kan hjelp være hensiktsmessig [33].

2.10.5 Dokumentasjon av resultater

Nøyaktig og konsekvent registrering av resultatene er avgjørende for å kunne bruke resultatene til fremtidig utvikling. Testere observerer brukervennlighetsproblemer under øktene og registrerer dem på problemlister eller dataloggen under, eller kort tid etter økten. Testeren registrerer observasjoner, for eksempel "så ikke alternativet", og tolkninger, for eksempel "forstår ikke grafikken". Testeren merker seg dersom samme problem dukker opp igjen. Når en oppgave utføres av flere brukere, jevnt, konsekvent og effektivt, må designet ha fungert Dette bør spesifikt merkes for å kunne bruke denne funksjonen som en mal i videre utvikling [33].

2.10.6 Analysering av data

Det er viktig å nøye analysere og forstå dataene som er samlet inn i testene, slik at utviklere best kan utnytte denne informasjonen i videreutvikling av produktet.

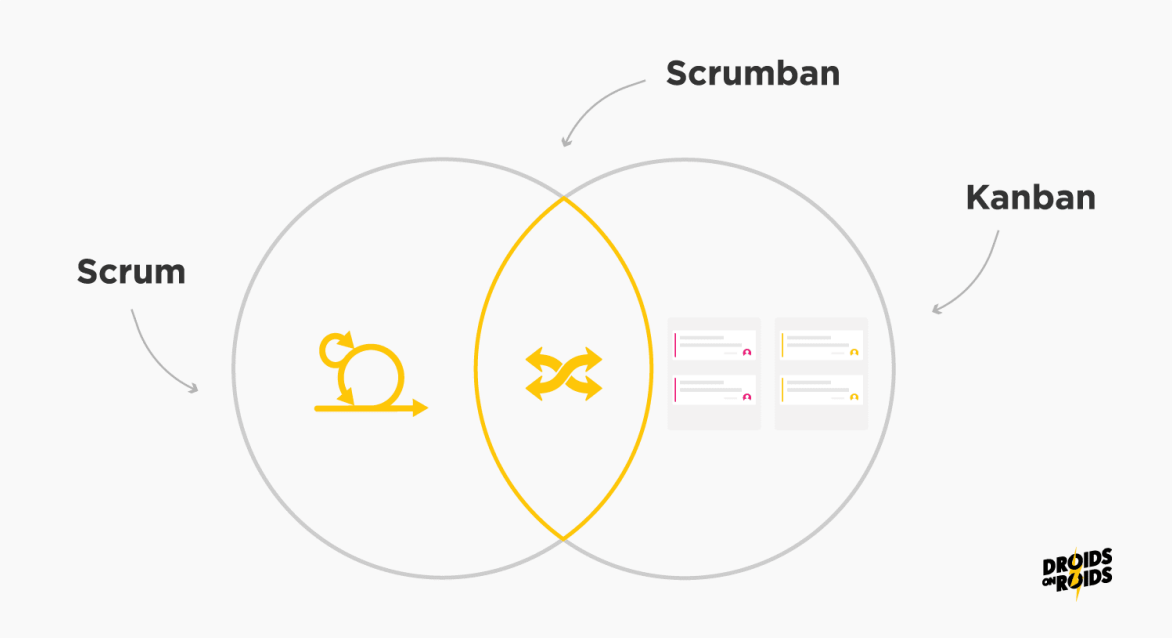
To hovedfokus legges på analysen av dataene fra brukertester: identifisering av kilden til problemet og tilordning av alvorlighetsgrad til problemer. Å identifisere roten til problemene som oppstår under testen gjør det enklere å rette opp i problemenes kilde i stedet for å lappe over dem. Dette tilrettelegger for bedre kodearkitektur. Å sette en

alvorlighetsgrad til hvert problem gjør det lettere å formidle hvilke problemer som bør fokuseres på først [33].

Når data er analysert er det viktig at de blir videreformidlet til de riktige mottakerene. Det har tidligere vært vanlig å gjøre opptak av testingen og fremlegge skriftlig rapport. Det er nå mer vanlig å ha en representant for utviklingsteamet sitte inne på noen av testene og holde et møte om resultatene etter [33].

2.11 Scrumban

Scrumban kombinerer elementer fra SCRUM og Kanban. En kanban-tavle blir brukt til å organisere oppgaver. Sprinter er valgfritt og SCRUM-seremonier blir gjort ved behov istedet for til faste tider. Scrumban egner seg for mindre team som vil benytte sprinter til å organisere arbeidet, da scrum er mest egnet for større team [2].



Figur 4: Scrumban kombinerer elementer fra SCRUM og Kanban [2]

2.11.1 Oppgavetavle

En Scrumban-tavle er basert på Kanban systemets tavleoppsett. Det er et visuelt verktøy som brukes for å få oversikt over arbeidsflyten og organisere oppgaver. Disse tavlene hjelper team med å organisere og spore arbeidsoppgaver gjennom hele prosjektet. Tavler

er vanligvis delt inn i kolonner som representerer ulike stadier eller faser i arbeidsflyten. Typiske kolonner inkluderer "To Do" (å gjøre), "Doing" (gjøres) og "Done" (ferdig). Arbeidsoppgavene representeres av kort. Hvert kort inneholder informasjon om oppgaven, som beskrivelse. Kortene flyttes fra kolonne til kolonne etter hvert som oppgavene går gjennom arbeidsflyten [34].

2.11.2 Sprinter eller kontinuerlig arbeid

Scrum setter både tids- og oppgavebegrensninger for hver sprint. Kanban derimot fokuserer på kontinuerlig arbeidsflyt. Scrumban er det mulig å jobbe med sprinter eller uten. Dersom det jobbes kontinuerlig må det settes begrensinger på hvor mange oppgaver som kan være aktive til en hver tid. Dersom teamet jobber i sprinter gjøres det begrensninger på hvor mange oppgaver som kan inngå i sprinten [35].

2.11.3 Ritualer

I Scrumban er det bare stand-ups som er et obligatorisk rituale, men team er fri til å innlemme de SCRUM ritualene som er nyttige i prosjektet. Andre møter inkluderer sprint planlegging, sprint review og retrospektiv [35].

Et stand-up-møte er et møte der hver av teamdeltagerne vanligvis står oppreist mens hvert medlem legger frem hva de har gjort dagen før, og hva de planleggere å gjøre den inneværende dagen. Dette gir alle i teamet oversikt over hva de andre holder på med og hindrer dermed at to personer jobber på det samme. Det gjør det også lett for en som har problemer å uttrykke det til teamet og få hjelp av en som kan mer om temaet [36]. Disse korte møtene er også en god måte å oppmuntre til teambygging og samhørighet, da de gir teammedlemmene litt mer sosialkontakt i arbeidstiden [35]. Grunnen til å stå oppreist under møte er å holde møtet kort og effektivt.

2.11.4 Sammenligning av SCRUM, Scrumban og Kanban

Tabellen under viser en sammenligning av elementer i Scrum, Kanban og Scrumban, som tydeliggjør hvilke elementer som er hentet fra hvilken av modellene for å lage Scrumban.

	SCRUM	**Kanban**	**Scrumban**
Regler	Strenge	Avslappede	Moderat strenge
Team størrelse	6 - 9 personer	Ubegrenset	Ubegrenset
Roller	Produkteier, Scrum master, utviklere, interessenter	Ingen spesifikke roller nødvendig	Ingen spesifikke roller nødvendig
Møter	Planlegging, sprint review, retrospektiv og daglig stand-up	Daglig stand-up	Daglig stand-up og andre møter ved behov
Arbeidssyklus	1 - 4 uker sprinter	Kontinuerlig arbeid	Kontinuerlig eller sprinter

Tabell 2: Sammenligning av SCRUM, Kanban og Scrumban [2]

3 Metode

3.1 Teknologivalg

3.1.1 Versjonskontroll

Det ble bestemt å bruke GitHub til versjonskontroll, slik at gruppen kunne ha prosjektet tilgjengelig for nedlasting selv etter avsluttede studier. GitHub er også den største Gitplatformen [37] og dermed det mest naturlige stedet for brukere å finne pakken. Gruppen kunne foretrukket å bruke GitLab, da denne platformen har bedre implementering av CICD og bedre verktøy for prosjektstyring. Å gjøre pakken som ble utviklet i oppgaven lett å finne for brukere samt å sikre langsiktig tilgang ble prioritert i valget av versjonskontroll.

3.1.1.1 Pakkeoppsett

Oppgaven innebar å utvikle en mellomvarepakke mellom radarsensor og Unity spillmotoren, referert til som pakken. Versjonskontroll ble bare gjort på pakken, og ikke et Unity prosjekt. Dette gjorde utvikling og bruk mer fleksibel igjennom prosjektet. En av grunnene til å bare ha versjonskontroll på pakken var at det er gjør det mulig for brukere å hente denne direkte fra GitHub, noe som er bedre for sluttproduktet i prosjektet. Fordi enkeltheten av å installere pakken var et fokus og noe av det som skulle brukertestes, ønsket gruppen at pakken skulle være tilgjengelig fra Git gjennom hele prosjektet. Dette gjorde brukertester lettere å gjennomføre. En av nedsidene ved å ikke organisere det som et Unity-prosjekt, i versjonskontroll er at man ikke får kjørt tester i CICD da disse må kjøres gjennom et prosjekt.

Det var klart at det ikke var ønskelig å levere koden som et prosjekt, siden det ikke er hensiktsmessig å tvinge brukere til å klone et helt prosjekt når de bare er ute etter pakken. I tillegg er det mye lettere å utvikle lokalt, dersom du er fri til å gjøre som du vil med prosjektet; at det ikke trenger å samsvare med prosjektet i versjonskontrollen. Det hadde imidlertid vært mulig å ha et prosjekt under utviklingen og ekstrahere pakken mot slutten av prosjektet. Dette hadde gjort det mulig å benytte CICD, selv om dette måtte blitt fjernet når pakken ble ekstrahert.

Siden det ble valgt å tilgjengeliggjøre importering av pakken gjennom Git er det spesielt viktig å ha en stabil hovedgren. Dersom en endring fører med seg ustabil kode inn i hovedgrenen vil alle brukere som har installert gjennom Git automatisk få denne ustabile koden.

3.1.2 Kommunikasjon via MQTT

Som beskrevet i seksjon 2.2 at godt etablerte pakker vil bidra til mer stabil og pålitelig kode. Disse ble derfor brukt der dette er relevant og godt egnet. Dette ble bestemt for å unngå å bruke tid på å kode egne løsninger der det allerede eksisterer gode implementasjoner, spesielt dersom dette var relativt urelevant for oppgaven. Videre er etablerte løsninger velutprøvde og relativt feilfri, og gjør det lettere å integrere prosjektet i andre prosjekter.

Et slikt område er implementasjonen av kommunikasjon via MQTT-protokollen, hvor vi har valgt å bruke en pakke som heter MQTTnet. Denne pakken gjorde det veldig enkelt å sette opp MQTT-kommunikasjon via server og klienter, og lot oss fokusere på brukervennligheten til pakken vår. Den er også en veletablert pakke til .NET-rammeverket som trolig er mer robust i bruk enn et server/klient-oppsett gruppen hadde utviklet selv.

For å ta i bruk MQTTnet var det nødvendig å finne ut hvordan NuGet pakker brukes i Unity prosjekter. Dette ble oppnådd ved å trekke ut den kompilerte koden til pakken i form av en dll-fil og redistribuere med koden til prosjektet. Dette var mulig siden MQTTnet er åpnen kildekode og derfor lovlig å distribuere.

3.1.3 API-kompabilitetsnivå i Unity

I prosjektet har vi valgt å støtte API-kompabilitetsnivået ".NET Standard 2.1". Dette er anbefalt fra Unity for å sørge for mest mulig støtte for kryssplattformsutvikling. Slik ble koden egnet til å kjøre på flest mulig typer enheter, slik som Linux, Windows og MacOS. Det skal være sømløst og enkelt å bruke pakken på alle plattformer slik at utviklere opplever minst mulig friksjon når de benytter seg av pakken. Videre er kompabilitetsnivået valgt siden dette tar liten plass, dersom noen vil kjøre koden på en mikrokontroller eller lignende plattformer hvor det er begrenset med lagringsplass eller andre ressurser.

Dette førte til at en enklere versjon av MQTTnet måtte brukes, men dette ble vurdert

som akseptabelt da versjonen som ble brukt støttet alle funksjonene gruppen hadde behov for. Det hadde også vært mulig å bruke en versjon som passet med en fullverdig versjon av denne pakken, men dette ville ikke hatt innvirking på pakken som ikke har behov for den ekstra funksjonaliteten denne versjonen tilbyr. Det ble derfor benyttet versjonen av pakken som støtter ".NET Standard 2.1"

3.1.4 Serialisering

Serialisering var sentralt i utviklingen da MQTT-protokollen sendte data i JSON-format. Det ble valgt å bruke Newtonsoft.Json-pakken som er tilgjengelig som en pakke i Unity. System.Text.Json er en raskere deserialiseringspakke, men denne ble ikke valgt da den ikke er tilgjengelig i Unity og dermed hadde måtte blitt redistribuert med pakken som tredjepartskode. System.Text.Json har også mange eksterne avhengigheter som også måtte ha blitt lagt til selv om gruppen bare hadde behov for enkle funksjoner fra pakken. Redistribuering av tredjepartskode var noe gruppen forsøkte å unngå da dette kan redusere annerkjennelsen til tredjepartskoden sin produsent. Et annet alternativ som er innebygd i Unity er JsonUtility-pakken, men dette var ikke mulig da denne ikke støtter deserialisering av C# sin datastruktur dictionary, som sensoren sender i sine meldinger. Det var ikke mulig å endre typene som sendes av sensoren.

3.1.5 Unity som utviklingsmiljø

I denne seksjonen vil det presenteres informasjon om diverse implementeringsmetoder som er direkte konsekvens av Unitys begrensninger eller beste praksiser.

3.1.5.1 Nullhåndtering

Siden Unity er sensitiv til nullhåndtering, var det viktig å være nøye med nullsjekker og å tillate få nullverdier. Der nullverdier var nødvendig måtte en spesiell merking (tag) legges til. Det er anbefalt i Unity å bruke disse så lite som mulig, så gruppen måtte vurdere nødvendigheten til å tillate nullverdier nøye.

3.1.5.2 Prefabrikerte spillobjekter

I Unity er det mulig å definere prefabrikerte spillobjekter (prefab) som er et ferdig laget spillobjekt som kan enkelt benyttes i spillverden. Manager-klassen ble implementert som en prefab for å gjøre det enkelt for brukere å sette opp denne.

3.1.5.3 Brukergenerert kode

Unity er en platform for utvikling og brukere av pakken vil lage skript som benytter og utvider funksjonenene som pakken tilbyr. Det var derfor viktig å vurdere hvilke funksjonaliteter som skulle innlemmes i pakken og hvilke som en bruker primært ville utvikle selv. Funksjonalitet gruppen forventet en bruker kunne ønske, men som trengte å spesialiseres for å få nytte av, ble implementert som eksempelkode slik at brukeren kunne få et eksempel på hvordan de kunne bruke pakken. Dette ville også gi bruker et eksempel på hvordan de kunne kode opp mot pakken.

3.1.5.4 UnityEvents

Det var viktig å frakoble koden fra bildeoppdatering for å unngå unødvendig ressursbruk. Observatør-mønsteret er svært nyttig i situasjoner der du vil oppnå høy kohesjon og lav kobling, siden du kan gi beskjed om oppdatering til en lang rekke med objekter uten at disse trenger å være tett sammenkoblet med objektet som sender ut data. Unity implementerer dette mønsteret via UnityEvents.

3.1.6 Kontinuerlig integrasjon og utrulling

Som beskrevet i visjonsdokumentet, vedlegg C, var det et mål at produktet skal være godt dokumentert. En del av god dokumentasjon innebærer at den må være oppdatert, som er automatiserbart via CICD. CICD-en til prosjektet ble konfigurert slik at dokumentasjonen ble automatisk generert fra spesielle kodekommentarer i kildekoden. Deretter ble den rullet ut på GitHub Pages, som er GitHub sin vertstjeneste for statiske nettsider.

3.1.7 Dokumentasjon

Doxygen ble brukt til å generere dokumentasjon fra spesielle kodekommentarer i kildekoden. Doxygen ble kjørt via CICD, og lagde en søkbar, statisk nettside med all informasjonen fra kommentarene som ble skrevet. Siden inneholder også mye informasjon om relasjonen mellom klasser og diverse Markdown-filer, slik som README-filen og filen som inneholdt lisensen til prosjektet. Doxygen ble også vurdert med tanke på WCAG. Den hadde god kontrast og var fult brukbar med tastatur. Dette ga gode resultater i universell utformingstester og ble derfor vurdert som en akseptabel generator.

Originalt var det ønskelig å utnytte samme verktøy som Unity til å generere dokumentasjonen til prosjeket; DocFX. Dette ville økt familiariteten til dokumentasjonen for de som allerede er godt kjent med Unity sin dokumentasjonstruktur, men var ikke mulig. DocFX krever kompilerte dll-filer å generere dokumentasjon, men på grunn av at det var en pakke som lå til grunn i versjonskontrollen, og ikke et Unity-prosjekt, var dll-filer opprettet i prosjektet ikke tilgjengelig i CICD. Det var derfor bedre å bruke en generator som ikke trengte kompilert kode.

I API-dokumentasjon ser man ofte eksempelkode som enkelt viser hvordan kan ta i bruk et API, og det samme gjelder for Unity. Det distribueres ofte kodeeksepmler sammen med pakker til Unity, slik at du kan se hvordan du tar i bruk pakken. Dette gjenspeiles også i Unity sin egen dokumentasjon.

3.1.8 Testing

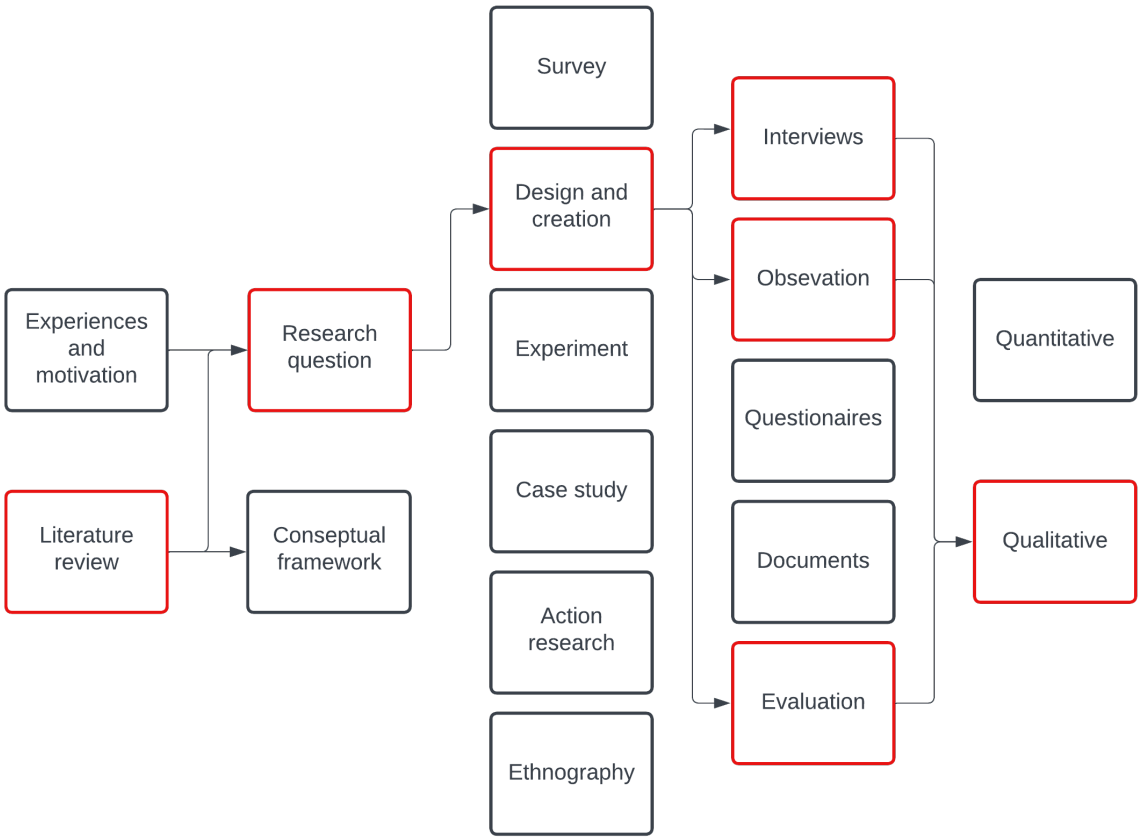
Kravene som ble stilt til testing i visjonsdokumentet var at produktet må være "godt testet", og at vi skal ha "høy testdekningsgrad der vi ser at dette er hensiktsmessig". Det ble derfor prioritert enhetstesting av grunnklassene i prosjektet.

Alle offentlige metoder som kunne testes uten simulering av sensorinndata ble skrevet enhettester for. Å lage simulering av sensorinndata ble vurdert til å være tidkrevende i forhold til nytteverdien i disse testene. Det hadde hvert mulig å lage en MQTT-klient som simulerer en sensor for å benytte i testing og under utviklingen av pakken. Dette ble ikke gjort da testnytten ble vurdert som lav.

3.2 Forskningsmetode

For å utforske problemstillingen [1.2] har vi valgt å benytte to metoder; litteraturstudie og brukertesting.

Ut i fra informasjonen fra litteraturen ble det utviklet funksjonalitet i mellomvarene som så ble brukertestet. Testene inkluderte observasjon av brukere under gjennomføring av oppgaver, intervju av bruker om deres opplevelse etter interaksjon med koden og evaluering av brukervennligheten etter testen. Hver av brukertestene ga kvalitativ informasjon om brukervennligheten til pakken. Dette er illustrert i figur 5:



Figur 5: Oversikt over vitenskapelig metode benyttet i prosjektet

Brukervennlighet er svært subjektivt noe som ville nødvendigjort brukertesting på en stor skala før kvantitative data kunne ha blitt ekstrapolert. Pakkens brukerbase er begrenset av tilgang til sensoren og erfaring med Unity ble dette ikke gjennomførbart [3.2.2.1]. Det var derfor bare hensiktsmessig å se på kvalitative resultater i denne oppgaven.

3.2.1 Litteraturstudie

I løpet av prosjektet gjorde gruppen en litteraturstudie om brukervennlighet. Det er få eller mulig ingen akademiske kilder om brukervennlighet for spesifikt Unity-pakker, så web API-er ble sett på som lignende nok til å gi nyttig informasjon. Programvarebibliotek hadde vært en mer direkte parallel, men litteratur om brukervennlighet for disse er også ikke lett tilgjengelig. Vurderingen av hvor sammenlignbart typen program var og mengden og kvaliteten til litteratur om temaet var grunnlaget for å velge å fokusere på web API-er. Sammenlignbarheten var lignende for begge, men API-er er langt mer grundig studert med tanke på brukervennlighet.

Det ble også lest om Kognitive Dimensjoner [2.9.4] som beskriver et rammeverk for brukervennlighet i notasjonssytemer som for eksempel programmeringsspråk. Dette ville gi innsyn i brukervennlighet for programvarebiblioteker og utviklingsmiljøer. Å ha kunnskap om brukervennligheten til pakkens kontekst, Unity og C#, vil gjøre det lettere å utbedre problemene og utnytte styrkene til disse.

3.2.2 Brukertester

Det ble utført en runde med brukertester omtrent ⅔ ut i utviklingsdelen av prosjektet og en runde med brukertester mot slutten av prosjektet. Disse ble gjort i henhold til kravene for gode brukervennlighetstester beskrevet i seksjon 2.10. Etter gjennomføring av den første brukertesten ble resultater analysert og benyttet som utgangspunk i neste utviklingssprint. Etter andre runde med brukertester ble mindre endringer gjort for å bedre problemer oppdaget testen.

Begge gruppemedlemmene satt inne på alle testene. Dette forsikret for gruppa at all erfaring fra testene var tilgjengelig for begge gruppemedlemmene. En av gruppemedlemmene stilte som testadministrator og en som referent. Hensikten med disse rollene var å redusere påvirkningen til gruppemedlemmene på resultatet av testen. Testadministrator fremla oppgavene til testpersonen og hjalp testpersonen med å overkomme problemer dersom nødvendig [2.10.4]. Referent dokumenterte brukervennlighetsproblemer som oppstod. Gode løsninger og positive tilbakemeldinger ble også dokumentert, i henhold til seksjon 2.10.5.

3.2.2.1 Testpersoner

For at testpersonene skulle være sluttbrukere eller potensielle sluttbrukere som beskrevet i seksjon 2.10.2, ble det bestemt å bare benytte personer med erfaring fra Unity. Dette på grunn av at gruppen Unity-utviklere ble sett på som sluttbrukere for pakken. Det ble ikke stilt noe krav til erfaring med sensoren eller MQTT-protokollen siden sluttbruker ikke kan forventes til å ha erfaring med dette.

3.2.2.2 Gjennomføring av testene

Før oppgavene ble testpersonen spurt om å vurdere sin egen erfaring med koding, C#, Unity og MQTT. Å ha forståelse for testpersonens erfaringsnivå synliggjør sammenhengen mellom erfaringsnivå og fremgangsmåte. En nybegynner vil for eksempel ikke være kjent med standard pakkestruktur og kanskje oppleve større problemer med å finne filene de er ute etter.

Deretter ble oppgaver gjennomført. Testadministrator kunne tilby hjelp eller hint dersom bruker sto fast, slik at testen kunne fortsette og ikke tok for mye av brukerens tid. Når testoppgaver ble laget ble det forsøkt å unngå å teste brukervennlighet som var begrenset av Unity sitt grensesnitt. Dette var ikke mulig å unngå helt, så testadministrator hjalp brukere fritt dersom de slet med noe som ikke var relevant til pakkens grensesnitt. Dette kunne være at enkelte brukere hadde problemer med å finne Unity sin pakkebehandler, og at andre brukere ikke klarte å lage 3D-objekter. Det ble valgt å hjelpe brukere med dette med en gang de slet for å redusere stress under testen og å ikke bruke tid på brukervennlighetsproblemer som ikke var del av prosjektet.

Etter oppgavene ble noen få spørsmål om brukerens opplevelse stilt. Dette var for å få konkret tilbakemelding på brukerens meninger om pakken. Dette kunne gi nytte da noen brukere mulig fremstår mer eller mindre fornøyd enn de virkelig er. I første omgang var disse spørsmålene: Hvordan synes du det var å sette opp pakken? Hvordan synes du det var å finne frem til skriptene du trengte? I andre omgang ble det spurt: Hvordan synes du det var å sette opp og konfigurere sensorer? Hvordan synes du det var å finne frem til skriptene og dokumentasjonen du trengte? I begge ble det spurt om brukerens helhetsopplevelse med pakken.

Under første runde med brukertester ble testpersonene nervøse av å kode foran andre og opplevde testen som en test av deres egenskaper. Dette presset ble forsøkt dempet i løpet

av den andre runden med tester ved å legge til denne beskjeden før testen:

"Dette er en test av brukervennligheten til pakken, ikke dine ferdigheter. Det er vanlig å bruke tid på å sette seg inn i et nytt kodebibliotek. Det er ikke meningen at alle oppgavene skal gjøres fult ut, i mange ønsker vi bare å se hvordan du oppsøker informasjon om koden."

I andre omgang ble det også reiterert at prosjektet hadde en README-fil og dokumentasjon, og at bruker var oppfordret til å ta seg god tid til å lese dette.

3.2.2.3 Oppgaver

Valg av oppgaver er beskrevet i seksjon 2.10.3 og fremlegger tre typer oppgaver som bør tas med i testen. Det var viktig at fokuset til testen var på brukeropplevelsen så spørmål stilt til bruker etter oppgaver var gjennomført relatere bare til opplevelsen av brukervennligheten.

I første runde med brukertester var det mulig å teste all funksjonaliteten i programmet, da programmet enda hadde få funksjoner. Oppgavene omfattet oppsett og enkel bruk av pakken. Fem oppgavescenarier [2.10.3] ble fremlagt:

1. "Du skal lage et Unity-prosjekt som benytter seg av SensMax TAC-B sensoren, og du vet at det finnes en pakke for dette. Implementer den i dette prosjektet."
   Denne oppgaven skulle teste oversiktligheten til GitHub-repoet og leseligheten til README-filen. Brukere ble vist GitHub-repoet. Etter første test ble brukere informert README-filen, da dette var noe en bruker ville ha sett mens de oppsøkte pakken som var en forutsettning for oppgaven.

2. "For å kunne kommunisere med sensoren må det settes opp administrasjon av MQTT-protokollen, fullfør oppsett av en manager og kjør programmet."
   Denne oppgaven testet både filstrukturen til pakken, og installasjonsveiledningen. En nybegynner kunne lese i README-filen og følge veiledningen der, mens en erfaren bruker kanskje ville sjekket prefab-mappen, da dette er hvor Unity-pakker som regel oppbevarer ferdig bygde objekter.

3. "Siden du får en feilmelding, ønsker du å feilsøke klienten. Endre manageren slik at klienten logger all aktivitet."
   Denne testet om brukeren fant og forsto innstillingene på manageren. Riktig gjennomføring av oppgaven over ville føre til en feilmelding for hver pakke fra sensoren

som var en forutsetting for denne oppgaven.

4. "Nå som manageren er klar ønsker du å håndtere innkommende meldinger fra sensoren med serienummer: 012345678. Legg til sensoren slik at den kan få pakker fra 012345678."
   Denne oppgaven sjekket om brukeren forsto hvor og hvordan sensorobjektet skulle legges til på manageren og gis riktige innstillinger.

5. "Sensor spillobjektet lagrer data fra sensoren. For å benytte denne i spillverden legg til et eksempelskript som lager spillobjekter som representerer folkene sensoren detekter på sensor objektet."
   Denne oppgaven testet om brukeren finner eksempelskriptene og forsto hvordan de skulle implementeres. Denne oppgaven ble endret underveis i testen da den opprinnelig ikke ba om et spesifikt skript og bruker ofte valgte et eksempelskript som krevde tre 3D-objekter, noe som tok mye tid å lage.

I andre runde med brukertester var målet å teste koding opp mot pakken. Denne testen forutsetter at pakken og sensoren er ferdig installert og konfigurert, slik at brukeren kan begynne å utforske koden direkte. Oppsett av pakke ble grundig testet i første omgang dermed ble det bestemt å ikke bruke tid på dette i denne testen. Til andre runde med brukertester ble disse oppgavene utarbeidet:

1. "Du har installert en pakke i Unity for å kommunisere med SensMax TAC-B sensoren. Du har lagt inn en manager som administrerer sensorer. Legg til en ny sensor til denne manageren."

   Denne oppgaven tester om brukere har lettere for å legge til sensor med knappen som ble lagt til etter forrige brukertest [4.2.1].

2. "Du har lyst til å ha brukernavn og passord på manageren og sensoren din. Sett opp manageren til å bruke brukernavnet "test" og passordet "test"."

   Denne oppgaven tester om brukeren forstår brukerveiledningen godt nok til å sette opp brukernavn og passord.

3. "Du ønsker å loggføre statistikk fra sensoren, og vet at den sender pakker med informasjon hvert 5. minutt. Legg til funksjonalitet for å loggføre disse pakkene."

   Dette tester igjen brukerveiledningens klarhet.

4. "Du tenker lage et skript som oppdaterer posisjonen til et objekt hver gang ny data mottas fra sensoren. Hvordan ville brukt pakken vår til å oppnå dette?"

   Denne oppgaven var ment til å teste hvor intuitiv den event-baserte arkitekturen er.

5. "Du skal lage et skript som bare benytter seg av objektet nærmest sensoren. Hvilken metode/funksjon ville du brukt?"

   Denne oppgaven var ment til å sjekke om brukeren finner enkelt metoder i dokumentasjonen/koden.

3.2.2.4 Analyse av testresultater

Etter testene hadde gruppen et møte der referatene ble godkjent av begge. Deretter ble hver oppgave rangert på en "enkelhetsskala". Enkelhetsskalaen var en nummerskala fra 1 til 5, der 1 var mest enkel og 5 var mest vanskelig. Problemer oppdaget under testen ble rangert på en alvorlighetsskala fra lav til høy. Denne vurderingen ga gruppen mulighet til å gjøre prioriteringer rundt ressursbruk på de forskjellige problemene. Det ble valgt å bruke enkle skalaer for vurdering, da dette bare skulle være en rask måte å prioritere oppgaver. Det hadde ikke vært nyttig å bruke mye tid på å finne en eksakt verdi for alvorlighetsgrad og vanskelighetsgrad for hver oppgave. Denne felles diskusjonen lot gruppemedlemene sammenligne erfaringer fra testene og komme til enighet om prioriteringer til neste sprint.

3.3 Utviklingsmetode

3.3.1 Prosesstyring

Scrumban ble benyttet som verktøy for å organisere arbeidet. Dette ble valgt da gruppen ønsket å jobbe ved hjelp av sprinter, men fant at SCRUMs strenge regler og mange ritualer ga for mye administrativ belastning for et team på to personer. Scrumban var mer egnet for et lite team da det bidrar til å organisere arbeidet uten å bruke for mye tid på ritualer. Kommunikasjon er lettere i små grupper og risiko for at noen blir overkjørt fordi de ikke uttrykker seg er lav. Gruppen kunne enkelt få diskutert hvordan en sprint gikk for eksempel uten at dette trengte å gjøres formelt for at alle i teamet skulle bli hørt.

Gantt-diagrammet fra forprosjektsplanen [B] ble brukt til å følge progresjonen. Ved endringer i planen ble Gantt-diagrammet oppdatert, slik at diagrammet til en hver tid viste

riktig tidslinje. Dette gjorde at gruppen tidlig oppdaget behov for endringer i planen, noe som var nyttig da Scrumban ikke implementerer tidsestimering av oppgaver. På grunn av dette var det ikke mulig å lage for eksempel et burndown-diagram.

Et Github-prosjekt ble brukt som kanban-tavle med kategoriene, "Backlog" (etterslep), "To Do" (å gjøre), "In progress" (i arbeid), "Review" (til godkjenning) og "Done" (utført). Oppgavene i todo var beregnet på inneværende sprinten, mens oppgavne i backlog var for senere sprinter.

Før en oppgave kunne sies å være ferdig, måtte den gjennomgås av det andre gruppemedlemmet. Dette ble implementert som en regel for sammenfletting på GitHub. Kode måtte godkjennes av det andre gruppemedlemmet før sammenflettingen kunne gjennomføres (se seksjon 3.3.5).

3.3.2 Møter med veileder og oppdragsgiver

Møter med oppdragsgiver ble holdt etter hver sprint og møter med veileder ble holdt omtrent hver tredje uke. Disse møtene var separate, selv om begge møtene var åpene for alle interessenter.

Møtene med oppdragsgiver hadde fokus på utvikling; både framdrift og resultater. På disse møtene fikk oppdragsgiver sett det foreløpige produktet. Det ble gitt tilbakemeldinger og etterspurt funksjoner Gruppen fikk også tips om Unity i disse møtene. Møtene med veileder fokuserte mer på fremgangen i prosjektet og spørsmål rundt kravene til oppgaven.

3.3.3 Sprinter

Prosjektet ble gjennomført i tre kodesprinter, som ledet opp til testing av programmet og en avsluttende sprint for å ferdigstille programmet. Oversikt over sprintene og testmetode kan sees i tabell 3. I starten av en sprint ble det planlagt en del oppgaver for sprinten, men oppgaver ble hovedsakelig lagt til underveis i sprinter. Dersom en oppgave ble oppdaget ble den lagt til i en passende sprint.

Sprint	**Lengde**	**Milepæl**
1	3 uker	Gjennomgang av MVP med oppdragsgiver
2	2 uker	Brukertester
3	3 uker	Brukertester
4	1 uke	Ferdigstilling av koden

Tabell 3: Oversikt over sprinter

3.3.4 Brukervennlighetsvurdering

I hvert steg av utviklingsprosessen ble brukervennligheten vurdert. Det var sentralt i oppgaven å lage et brukervennlig produkt, og som beskrevet i seksjon 2.9 er det vist at å fokusere på brukervennlighet tidlig i utviklingsprosessen reduserer brukervennlighetsproblemer i sluttproduktet og før brukertester. Mest brukervennlig implementasjon av en ny funksjon ble drøftet før utviklingen av en ny funksjon startet. Som del av kodegjennomgang ble brukervennligheten vurdert før koden ble sammenflettet inn i hovedgrenen.

3.3.5 Kodegjennomgang

Før sammenfletting inn i hovedgrenen på GitHub kunne utføres, måtte koden gjennomgås. Det var flere krav til kode i disse gjennomgangene. For å kunne godkjennes måtte kode være fult dokumentert og enhetstester være skrevet der det var mulighet for dette. Det var også som nevnt over en forventning til brukervennligheten til den implementerte funksjonen.

4 Resultater

4.1 Ingeniørfaglige resultater

Det ble utviklet en pakke til Unity som fasiliterer kommunikasjon med SensMax TAC-B mennesketellende radarsensor. Programmet som ble laget er modulært for at brukere enkelt skal kunne implementere bare den funksjonaliteten de har behov for. Programmet ble implementert som en Unity pakke for å gjøre det mulig for brukere å enkelt installere den fra GitHub. Det er to skript som er sentrale i pakken; Manager og Sensor.

Manager håndterer MQTT-kommunikasjon mellom Unity, og sensorene som er koblet til. Manager-klassen implementerer både MQTT-server og -klient. Serveren tar i mot pakker fra sensorer og videreformidler den til klienten som pakker ut informasjonen og videreformidler denne til det rette sensorobjektet. Manager opprettholder også en liste med sensorobjekter som representerer hver sensor koblet til brukers prosjekt.

Sensor-klassen er en representasjon av den fysiske sensoren i spillverdenen. Den håndterer dataene motatt fra en spesifikk sensor. Den opprettholder en liste av alle objektene som sensoren detekterer som brukeren kan benytte i sine prosjekter. En sensor kan være "aktiv" dersom den detekterer objekter eller "inaktiv" dersom ingen objekter blir detektert. Sensorens nåværende og forrige aktivstatus, når denne statusen begynte og hvor lenge den har vart blir lagret i sensor objektet. Dette kan benyttes av brukeren til å føre statistikk. Sensoren implementerer UnityEvents for at brukere kan slippe å oppdatere objekters posisjon etter hver bildeoppdatering. Dette gjør det mulig å lytte til eventen å oppdatere posisjoner bare når en ny melding kommer fra den fysiske sensoren.

4.1.1 Produktets funksjonelle krav

I visjonsdokumentet, vedlegg C, er det diskutert en rekke krav som er en naturlig videreutvikling av de kravene som ble stilt av oppdragsgiver. Den originale kravspesifikasjonen fra oppdragsgiver blir fremlagt i vedlegg A.

Tabell 4: De funksjonelle kravene til produktet

Krav	**Oppfylt**	**Kommentar**
MQTT-server som kommuniserer mellom Unity og sensoren	×	
Pakken kan bli installert i alle typer Unity-prosjekt	×	"Alle typer" tolkes her som i både 2D og 3D prosjekter. Dette er støttet, men krever litt omregning for å brukes i 2D-prosjekter.
Funksjonalitet for å konvertere koordinat-data fra sensoren til koordinat-data som kan brukes i Unity, for eksempel til å posisjonere spill-objekter som spillerens avatar	×	Ingen konvertering nødvendig da dette er en innebygd funksjon i Unity
Funksjonalitet for å lese bevegelses-data fra sensoren	×	"Bevegelsesdata" tolkes her som hastighetsdata i tillegg til posisjons-data
Server starter når Unity prosjektet starter	×	
Mulighet for å kommunisere med flere sensorer samtidig	×	
Funksjonalitet for å registrere sensorens plassering i spillmiljøet med id	×	
Funksjonalitet for å bestemme hva pakken skal lese av informasjon fra en gitt sensor (posisjonsdata, bevegelsesdata, begge)		Ikke hensiktsmessig. Siden begge datatypene sendes med sensormeldingene og lagres kan bruker selv velge hva de ønsker å benytte.
Funksjonalitet for å bare få informasjon om det er en person i en gitt sone fra sensor		Sonefunksjonalitet ble ikke utforsket

Fortsettelse av tabell 4		
Krav	**Oppfylt**	**Kommentar**
Funksjonalitet for overvåkning av hvor mange mennesker som er i et bygg/rom ved hjelp av en eller flere sensorer	×	Må programmeres av bruker, men er støttet
Støtte for begge størrelsene pakke sensoren kan sende og automatisk deteksjon dette	×	

4.1.2 Produktets ikke-funksjonelle krav

Visjonsdokumentet, vedlegg C, fremlegger ikke funksjonelle krav til dokumentasjon, brukervennlighet og pålitelighet. Under vil det som inngår i de ikke funksjonelle kravne bli beskrevet i mer detalj.

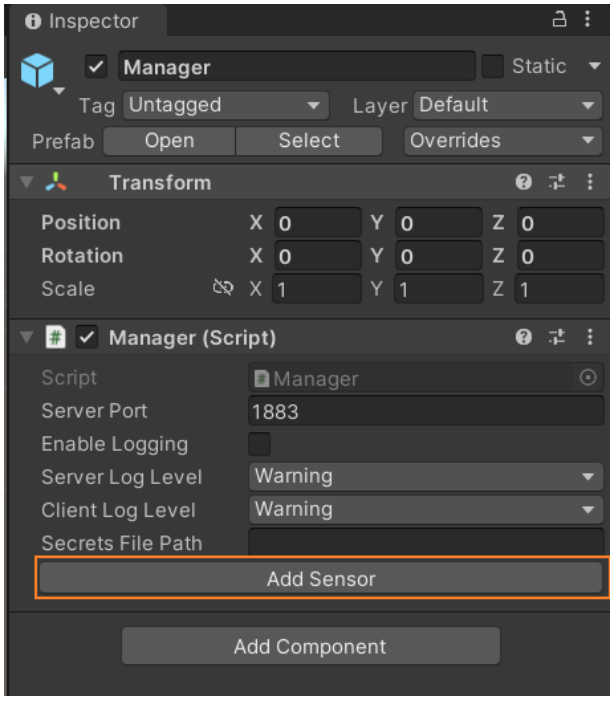
4.1.2.1 Dokumentasjon

Visjonsdokumentet fastsatte mål om fult dokumentert kode. Det er derfor derfor skrevet grunding, utfyllende dokumentasjon i koden både i form at dokstrenger og Unity tips som er synelig i Unity sitt grensesnitt. Det er også formulert en README-fil som uthever de viktigste funksjonene til pakken, slik at man raskt kan sette seg inn i funksjonaliteten til systemet.

4.1.2.2 Brukergrensesnitt

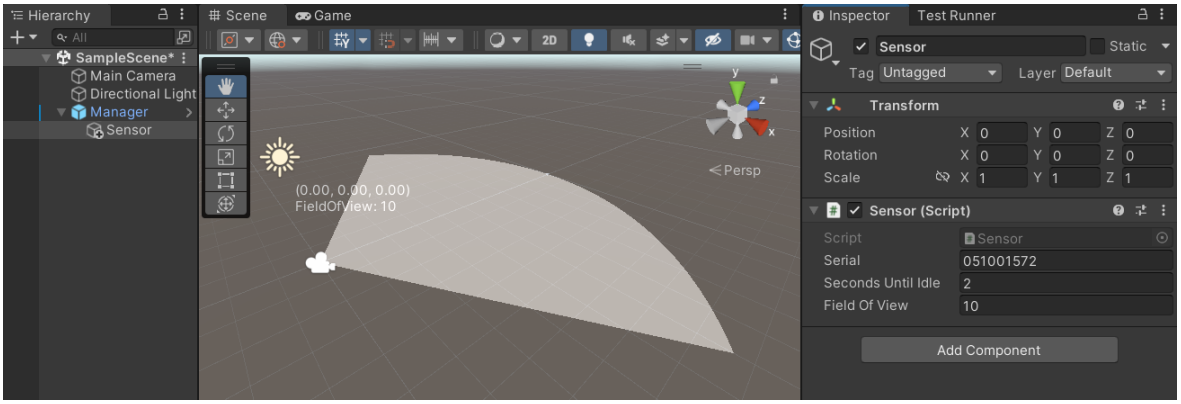
I visjonsdokumentet ble det bestemt at brukergrensesnittet skulle være “klart, tydelig og enkelt å forstå”. Dette ble gjennomført på to måter, ved å følge Unitys beste praksiser for pakkestruktur og å lage utvidelser av Unity sitt eksisterende brukergrensesnitt. Det ble blant laget annet en knapp slik at brukere kan enkelt legge til flere sensorer. Denne er merket i figur 6.

Den fysiske sensoren har et synsfelt som er 120°, med en radius på 10 meter. For å representere dette i spillverdenen slik at brukere kan orientere sensorobjektet riktig ved



Figur 6: Manager i Unity Editor med knapp merket

behov, ble det laget en visuell representasjon av synsfeltet til sensoren som vises i redigeringsmodus i Unity.



Figur 7: Sensor i redigeringsmodus i Unity

4.1.2.3 Feilsøking

Produktet skulle også være responsivt, i den form at det gir beskjed ved feil og mangler. Vi har implementert dette via Unity sin konsoll, som printer diverse informasjon dersom det for eksempel oppdages feil. Det er mulig for brukeren å justere hvilken grad av informasjon

de ønsker ved hjelp av justering av loggnivå. Dette gir en bruker god mulighet til å feilsøke både server og klient individuelt slik at bruker ikke blir overveldet av uønsket informasjon.

4.1.2.4 Brukseksempel

For at produktet skulle være lett å ta i bruk for nye brukere ble det utviklet eksempelkode som viser hvordan en bruker kan benytte programmet. Et av målene fra forprosjektsplanen [B] var å ha minst et eksempel på hvordan pakken kan brukes i Unity-prosjekter. Det ble laget flere eksempler på skript for sensorobjektet som brukere kan bruke som utgangspunkt når de skal bruke pakken i sine prosjekt. Disse er ment til å være både eksempler på potensielle bruksområder og på hvordan bruker kan kode sine egne skript for sensorobjektet.

4.1.2.5 Utvidet funksjonalitet

I visjonsdokumentet fremlegges kravet "produktet skal være modulært og tilby bruker muligheten til å legge til ikke essensiell funksjonalitet bare ved behov." Pakken ble implementert slik at utvidelser av funksjonalitet var enkelt. Dersom en bruker bare vil bruke enkle basisfunksjoner har de mulighet til dette, men brukere som ønsker utvidet funksjonalitet kan enkelt legge til dette gjennom Unity sitt grensesnitt. Siden managerklassen ikke skal endres av brukeren var det viktig å holde denne enkel og effektiv. For at brukeren bare skal få nødvendig funksjonalitet som utgangspunkt, ble utvidelser som er valgfrie lagt til som separate skript som ved behov kan legges til i manager. Det ble laget et slik skript i prosjektet som loggfører 5 minuttermeldingene sensoren sender til en fil. Dette er en egen MQTT-klient som mottar og håndterer disse pakkene.

4.1.2.6 Universell utforming

Det ble etablert i visjonsdokumentet at vi skulle følge Web Content Accessibility Guidelines (WCAG) der det var hensiktsmessig, men da vi utviklet få nettsider ble ikke dette så relevant som opprinnelig antatt. Alle brukegrensesnittene til prosjektet var generert av etablerte aktører som har innebygde universell utformingstiltak. README-filen blir vist gjennom GitHub sitt grensesnitt, Unity Editor har muligheter for å endre grensesnittet til bruker behov og Doxygen generer nettsider som er i samsvar med WCAG.

4.1.2.7 Installasjon

Det var også et mål å gjøre pakken lett å installere da en vanskelig installasjonsprosess vil avskrekke potensiselle brukere. Derfor ble det tatt hensyn til hvordan avhengigheter påvirker installasjonen. MQTTnet ble distribuert med pakken. Dette gjør at brukeren ikke trenger å legge til dette selv. Pakken benytter Newtonsoft.Json, men dette er en pakke tilgjengelig gjennom Unity og kunne legges til som en avhengighet istedet for å redistribueres. Brukeren får fortsatt automatisk satt opp avhengigheten gjennom Unity.

4.1.2.8 Pålitelighet

Det legges frem krav til pålitelighet i visjonsdokumentet, vedlegg C. Produktet må ha godt testet og verifisert pålitetlighet, da det skal kjøre over lengre tid uten oppsyn. Uten mulighet til å teste produktet over lengre tid grunnet kontinuerlig utvikling og korte sprinter, var det hensiktsmessig å sørge for pålitelighet ved å følge diverse kodestandarder og å teste koden grunding.

4.1.2.9 Testdekning

Det ferdige produktet har en testdekning på 60% i form av enhetstester for å sikre et stabilt produkt. Testene er skrevet for Unity Test Runner.

4.1.2.10 Kryssplatformsstøtte

Ved å bruke den anbefalte .NET-versjonen til Unity ble det sikret at pakken var brukbar på de fleste operativsystemer. Dette inkluderer også mikrokontrollere der lagringsplass er svært begrenset. Et av målene lagt frem i visjonsdokumentet var at pakken skulle være tilgjengelig på alle platformene Unity legger tilrette for.

4.1.2.11 Langsiktig vedlikehold

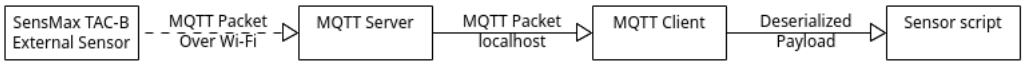
For at pakken skulle være pålitelig på langsikt med et lavt behov for vedlikehold, ble det prioritert å bruke versjoner av avhengigheter som er langsiktig vedlikeholdt. Siden pakken

ikke kan garantere langsiktig vedlikehold var det et mål å redusere behovet for fremtidig vedlikehold.

Et mål var å lage kildekode som er lett å videreutvikle. Det ble vektlagt å følge prinsipper for ren kode, som fokuserer på å skrive kode som er lesbar og godt strukturert. Hensikten er å gjøre det enkelt for andre å forstå og forandre koden ved behov. Videre var det viktig å begrense antallet eksterne avhengigheter for å gjøre det enkelt å oppdatere og vedlikeholde pakken. Slik blir det mindre sannsynlig for oppdateringer til eksterne avhengigheter å påvirke pakken på en negativ måte. Koden skal distribueres som åpen kildekode med en ettergivende lisens for å oppfordre til videreutvikling og eksperimentering. Dette har blitt gjennomført, og koden ligger åpent ute på GitHub med en medgjørlige MIT-lisens.

4.1.2.12 Sikkerhet

I visjonsdokumentet ble det fremlagt et mål om å ikke innføre sikkerhetshull som ikke alerede eksisterer i Unity og MQTTnet. Dette målet ble nådd da MQTTnet håndterer overføring av data mellom sensor og manager. Resten av kommunikasjonen internt i pakken skjer lokalt og er dermed så sikkert som konteksten det eksisterer. Pakkeflyten i prosjektet er illustrert i figur 8. Det ble imidlertid ikke implementert kryptering av pakkene mellom sensor og manager, slik at disse pakkene er mulige for en angriper å hente ut. Pakkene inneholder ikke sensitiv personinformasjon og er derfor ikke kritiske å kryptere.



Figur 8: MQTT-pakke flyt for UniTac

4.2 Vitenskapelige resultater

Under vil resultatene fra brukertestene legges frem. I spørsmålsdelene, og der brukere hadde forskjellige tilnærminger, ble det nødvendig å skille mellom brukere. Brukere blir da referert til med fornavn, med mindre bruker ønsket å forbli anonym.

4.2.1 Første omgang med brukertester

Detaljerte resultater fra første runde med brukertester kan leses i vedlegg D.1. Under vil en analyse av disse resultatene oppgis.

Oppgave 1: Brukeren skulle legge til pakken i et Unity-prosjekt. I denne oppgaven støtte mange av brukerene på et problem med å bruke SSH til å installere pakken fra GitHub. Etter brukertestene ble denne oppgaven vurdert til å være nivå 2 på enkelhetskalaen. De fleste brukerene utførte oppgaven på riktig måte ved hjelp av README-filen, på tross av eksterne tekniske problemer. Unntaket var første brukertest, der bruker ikke oppsøkte README-filen da han antok at den ikke eksisterte. I de følgene testene ble brukerne informert om README-filen før de begynte.

Oppgave 2: Bruker skulle legge til en manager i spillverden. Alle brukerene fikk til denne oppgaven uten problemer. Etter brukertestene ble denne oppgaven vurdert til å være nivå 1 på enkelhetskalaen. Prefab av manager gjorde denne oppgaven veldig enkel. Løsningen ble vurdert som funksjonell og god.

Oppgave 3: I oppgaven skulle bruker endre loggføringsnivået i manager-objektet. Alle testdeltakerene fikk til denne oppgaven helt uten problemer. Etter brukertestene ble denne oppgaven vurdert til å være nivå 1 på enkelhetskalaen.

Oppgave 4: Brukere skulle legge til en sensor og gi den serienummer. Alle brukerene la til serienummeret uten problemer, men de fleste brukerene hadde problemer med å legge til sensorenen under manageren. Dette var det største brukervennlighetsproblemet oppdaget i den første brukertesten. Med mindre brukeren leste README-filen veldig nøye, la ikke brukeren til sensor objektet under manager. Det ble derfor oppdaget at README-filen måtte tydeliggjøres for å hjelpe brukeren. Etter brukertestene ble denne oppgaven vurdert til å være nivå 5 på enkelhetskalaen.

Oppgave 5: I denne oppgaven skulle et eksempelskript legges til på sensoren. Alle

brukerene fikk til oppgaven, men mange hadde problemer med å finne riktig fil. De fleste måtte sjekke README-filen to ganger for å finne samples-mappen. Det var også mange som hadde problemer med å lage 3D-objekter i Unity, men dette stammet fra brukergrensesnittet til Unity, ikke denne pakken. Etter brukertestene ble denne oppgaven vurdert til å være nivå 4 på enkelhetskalaen.

Spørsmål 1: Hvordan synes du det var å sette opp pakken?

Bruker	**Svar**
Fredrik	Foretrekker å importere pakker som assets fra release. Synes alltid det er litt rotete å importere pakker. Etterspør flere muligheter for importering.
Anonym	Opplevde litt problemer med Git. Ville nok klart å finne ut av det selv, men synes det var ganske vanskelig.
Felix	Helt fint, å legge til pakken med Git-url virker som en bra standard.
Tore	Veldig greit, godt forklart av dokumentasjonen. Ville ha satt seg mer inn i skriptene hvis han faktisk skulle jobbe med pakken.
Petter	Veldig greit, enkelt å bruke Unity sin innebygde installasjons metode. Likte at det var satt opp prefabs, og at det kan legges til flere sensorer. At han fikk opp i konsollen hvilket serienummer som manglet som gjorde det lett å registrere sensor. Føltes lett å få opp å kjøre.

Tabell 5: Brukeres tanker om enkelheten til oppsett av pakken

Spørsmål 2: Hvordan synes du det var å finne frem til skriptene du trengte?

Bruker	**Svar**
Fredrik	Synes sensor/manager relasjonen var litt forvirrende. Ønsker at sensorer blir generert automatisk.
Anonym	Synes det gikk veldig fint å finne frem.
Felix	Fant fort til å ikke være vant til Unity. Synes pakkestrukturen er fin.
Tore	Det var greit med README, men synes det er rart at samples lå i run-time.
Petter	Gikk greit, måtte bare bli kjent med strukturen. Men gikk veldig fort.

Tabell 6: Brukeres tanker om filstrukturen til pakken

Spørsmål 3: Hvordan var helhetsopplevelsen av pakken?

Bruker	**Svar**
Fredrik	Syns det var greit, likte godt eksempelkoden.
Anonym	Opplevde pakken som bra. Synes han ble litt påvirka i positiv retting av at han fikk hjelp, kunne ha likt den mindre hvis han ble stående fast på en oppgave lengre.
Felix	Ganske bra, virket enkelt å bruke. Eneste problemene han opplevde var med Unity.
Tore	Virka grei, har ikke testet påliteligheten. Virka relativt enkelt å bruke til å styre spill objekter.
Petter	Veldig bra og nyttig. Følte han kunne fått oversikt selv om han hadde satt seg inn i pakken alene.

Tabell 7: Brukeres helhetsopplevelse i første brukertest

Problemer oppdaget i løpet av denne testen var:

- Å ikke ha release av en unitypackage tilgjengelig begrenser hvilke måter pakken kan implementeres.
  Alvorlighetsgrad: Lav
- Å legge til sensorobjektet som barn av manager var lite intuitivt og ble opplevd som tungvindt.
  Alvorlighetsgrad: Høy
- Å finne eksempelkoden i mappestrukturen var vanskeligere en nødvending.
  Alvorlighetsgrad: Medium

4.2.2 Andre omgang med brukertester

Detaljerte resultater fra første runde med brukertester kan leses i vedlegg D.2. Under vil en analyse av disse resultatene oppgis.

Oppgave 1: I denne oppgaven skulle bruker legge til en ny sensor på manager. Dette skulle fortrinnsvis gjøres ved en knapp, men det var alternative løsninger. Alle deltakerene fikk til oppgaven uten problemer når de hadde lest dokumentasjonen, men knappen var ikke synlig nok til å sees uten å lese dokumentasjonen. Bare Rinsai som primert hadde sin kode erfaring fra Unity så knappen med engang. Derfor ble denne oppgaven likevel vurdert til å være nivå 3 på enkelhetsskalaen. Lite synlig knapp ble notert som et brukervennlighetsproblem.

Oppgave 2: Brukeren skulle konfigurere brukernavn og passord på MQTT-protokollen. I denne oppgaven hadde alle brukerene problemer med å forstå dokumentasjonen. Brukere måtte lage en tekstfil i prosjektet. Dette kunne ikke gjøres gjennom Unity, så brukere måtte åpne filutforsker for å gjøre oppgaven. Det var ikke klart for brukerene hvilket filformat de skulle bruke eller hvor de skulle lage filen. Etter brukertestene ble denne oppgaven vurdert til å være nivå 5 på enkelhetskalaen.

Oppgave 3: I denne oppgaven skulle brukeren finne og legge til et skript på manager-objektet. Alle brukerene i denne testen fikk til oppgaven, men det var ikke klart gjennom oppgaven at å lage en fil for å lagre dataene til skriptet var valgfritt. Dermed brukte mange ekstra tid på å lage fil, med samme problemer som i oppgaven før. Etter brukertestene ble denne oppgaven vurdert til å være nivå 2 på enkelhetsskalaen.

Oppgave 4: Brukeren skulle finne en spesifikk event i sensor-skripet. Brukere lette lenge i kodebasen før de fant eventen. At den event-baserte arkitekturen ikke var forklart i README-filen var et brukervennlighetsproblem. Etter brukertestene ble denne oppgaven vurdert til å være nivå 5 på enkelhetsskalaen. De forskjellige fremgangsmåtene brukt kan sees i tabell 8.

Bruker	**Fremgangsmåte**
Aleksander	Etter en del leting fant han en annen event enn det som var etterspurt.
Peder	Fant eventen som ble etterspurt ved å se gjennom sensor-skriptet.
Anonym	Fant riktig event ved å se gjennom eksempelkoden, men kommenterte at det ikke var klart om eventen sender data eller ikke.
Matilde	Fant eventen etter leting i sensor-skriptet, men var usikker på hvordan eventer fungerer i Unity.
Trym	Kommenterte at eventen kanskje var løsningen etter å ha gitt en annen løsning.
Rinsai	Lette med engang etter en event, men fant den ikke i Unity Editor grensesnittet så bestemte seg for at det måtte være noe annet. Fant eventen relativt raskt i koden etter oppklaring av oppgaven.

Tabell 8: Brukerene fremgangsmåte for å finne event i koden

Oppgave 5: Bruker ble bedt om å finne metoden med en gitt funksjon. Det er generelt vanskelig å finne en gitt metode i en kodebase, så det ble tatt med i vurderingen av denne oppgaven. Den ble vurdert til å være nivå 1 på enkelhetsskalaen. Alle fant riktig metode eller la frem en alternativ løsning relativt enkelt. I denne oppgaven åpnet de fleste brukerene dokumentasjons siden, men ingen valgte å gå videre med å se på dokumentasjonen i dette formatet. En bruker kommenterte på at utseende til dokumentasjonssiden var gammeldags og uinnbydende.

Spørsmål 1: Hvordan synes du det var å sette opp og konfigurere sensorer?

Bruker	**Svar**
Aleksander	Følte at maglende erfaring med Unity var en stor utfordring, men uttrykte at dersom han hadde hatt tid og ingen som så på og dokumentasjonen skulle han fått oversikt etterhvert.
Peder	Synes testen ikke ga han en forståelse for hva pakken var ment til, som gjorde at han ikke kunne ha en klar mening om den. Synes forklaringen av hvordan man satte opp brukernavn/passord var uklar på hvordan filen skulle henvises til.
Anonym	Synes å legge til sensorer både med knapp og manuelt var greit. Synes å opprette ekstern fil for brukernavn/passord var tungvint, ville helst hatt felter i Unity.
Matilde	Syns det var veldig lett å legge til nye sensorer.
Trym	Synes knappen gjorde det veldig lett å legge til sensorer.
Rinsai	Enkelt. Ingen problemer. Lurte bare på filformatet til loggingen.

Tabell 9: Brukers opplevelse av sensoroppsett

Spørsmål 2: Hvordan synes du det var å finne frem til skriptene og dokumentasjonen du trengte?

Bruker	**Svar**
Aleksander	Søkemenyen var bra. Dokumentasjonen var jo bra.
Peder	Veldig enkelt å finne fram til skriptene og dokumentasjonen han trengte.
Anonym	Opplevde dokumentasjonsnettsiden som svært uinnbydende, ville ikke ha benyttet den. Etterspurte flere bilder i README-filen som viste oppsett da rent skriftlige forklaringer er vanskelige å følge.
Matilde	Å finne frem til skriptene var veldig greit, de var godt dokumentert som gjorde å finne metoder lettere. Var litt usikker på hva som inngikk i manager og i sensor.
Trym	Synes det var greit og trodde han hadde synes det var lettere hvis han hadde brukt Unity mer nylig. Hadde hjulpet å vite mer om hvordan pakken ble laget. Generelt tydelig README, med bare noen uklarheter.
Rinsai	Fant fram til skript. Ville egentlig finne metoden etterspurt i oppgave 5 på dokumentasjonsnettsiden, men følte ikke hun kunne bruke denne siden hun hadde fått inntrykk av at denne ikke inngikk i testen.

Tabell 10: Brukeren opplevelse av filstrukturen og dokumentasjonen

Spørsmål 3: Hvordan var helhetsopplevelsen av pakken?

Bruker	**Svar**
Aleksander	Følte ikke han hadde fått nok innsikt i pakken til å ha noe helhetsinntrykk.
Peder	Vikret veldig ryddig, bra laget og lett å få opp å kjøre. Godt dokumentert kode. Likte at pakken inneholder samples, tests og de fleste andre ting han ønsker i en Unity-pakke.
Anonym	Virket gjennomført og enkel å implementere. Liker eksempelkoden, var fint med et godt eksempel på bruk.
Matilde	Veldig grei. Var forskjellig fra det hun bruker Unity til da hun ikke hadde erfaring med sensorer. Tok litt tid å sette seg inn i, men lett å få forståelse på grunn av god dokumentasjon.
Trym	Veldig enkel å bruke. Enkel og forståelig kode. Følte det var veldig brukervennlig.
Rinsai	Bra, hadde ikke mye kritisere med pakken. Etterspurte dokumentasjon i Unity sin brukergrensesnitt.

Tabell 11: Brukerens helhetsinntrykk av pakken

Problemer oppdaget i løpet av denne testen var:

- Knappen for å legge til sensor var ikke synlig nok i Unity sitt grensesnitt.
  Alvorlighetsgrad: Medium
- Det ble ikke tydelig lagt frem for bruker hvilken filtype filer som skulle legges til skulle være.
  Alvorlighetsgrad: Høy
- Brukere fikk ikke nok informasjon om når filer ville automatisk genereres og gjorde dermed unødvendig arbeid for å legge til disse.
  Alvorlighetsgrad: Lav
- Det var ikke tydelig gjennom dokumentasjonen hvilke events pakken benytter eller at eventer ble benyttet.
  Alvorlighetsgrad: Høy
- Mangel på dokumentasjon som er synlig i Unity Editor begrenser brukere som primært bruker denne funksjonen.
  Alvorlighetsgrad: Medium

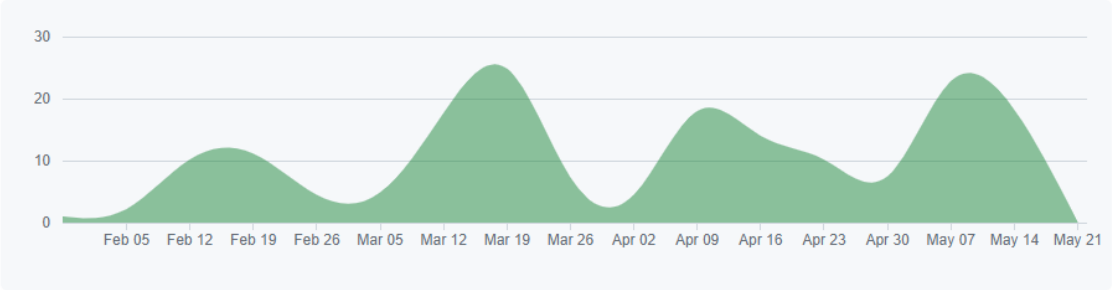
- Dokumentasjonsnettsiden var visuelt uinnbydende for mange brukere som dermed valgte å se bort i fra den.
  Alvorlighetsgrad: Lav

4.3 Administrative resultater

4.3.1 Fremdriftsplan

Det ble opprinnelig planlagt å ha tre utviklingssprinter, en runde med brukertester og to runder med utforskning av bruksområder for sensoren i Unity [B]. Det ble gjort store endringer i denne planen. Det ble gjennomført tre utviklingssprinter, og to runder med brukertester. Utforskning av bruksområder for sensoren ble ikke gjennomført.

Et gantt-diagram som viser den faktisk gjennomførte prosjekt planen kan sees i seksjon B.2. Det opprinnelige gantt-diagrammet som ble vedlagt forprosjektsplanen er også vedlagt [B.1] for sammenligning. Det ble i forprosjektsplanen lagt frem en ide om å ville utforske bruksområder for sensoren, dette ble ikke gjort da det ikke ville fremmet utforskningen av problemstillingen. De fire kodesprintene kan sees i oversikten over commits fra GitHub i figur 9.

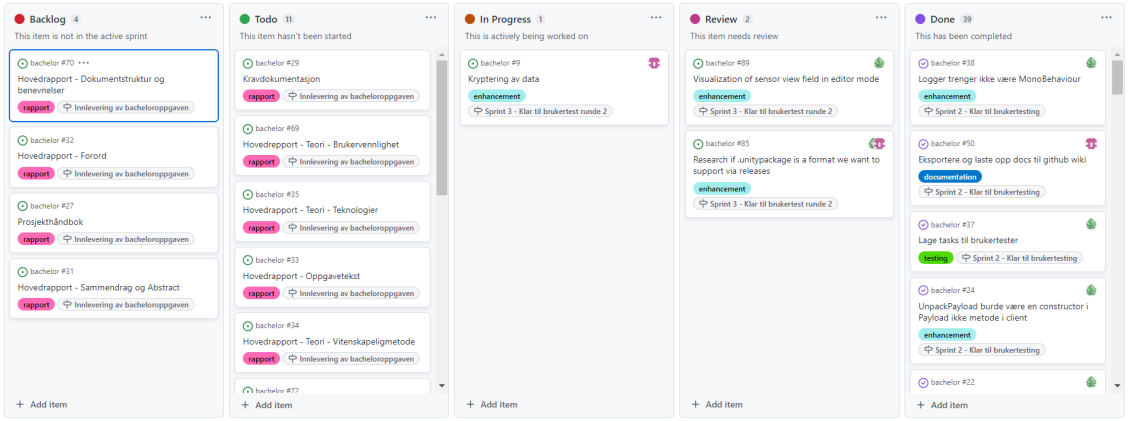


Figur 9: Oversikt over kode lagt til gjennom Git

4.3.2 Arbeidsprosessen

Som beskrevet over ble det gjennomført fire utviklingssprinter. Disse ble organisert rundt milepæler. Tidlig i prosjektet ble det fordelt oppgaver til hver sprint, oppgaver oppdaget i løpet av prosessen ble lagt til i en passende sprint. Etter hver milepæl ble et uformelt retrospektiv gjennomført i gruppen. Etter dette ble oppgaver for neste sprint planlagt.

Gruppen brukte GitHub issues som en Scrumban-tavle til å organisere arbeidet i løpet av sprintene.



Figur 10: Scrumban-tavle brukt av gruppen under prosjektet

Kodegjennomgang ble implementert som en regel på GitHub, slik at all kode ble gjennomgått før sammenfletting med hovedgrenen.

5 Diskusjon

5.1 Diskusjon av ingeniørfaglige resultater

Utvikling i og for Unity tilbyr en helt spesiell kontekst. Det er en høyst spesialisert programvare som enklere lar deg utvikle 3D-applikasjoner, men som også medbringer en rekke utfordringer og unike måter å gjøre ting på.

Som en godt etablert programvare har Unity etterhvert utviklet solid dokumentasjon som videreformidler deres beste praksiser og standarder. Unity sine fordeler, ulemper og standarder har hatt stor innvirkning på hvordan vi har utviklet, og dermed også stor innflytelse på det endelige resultatet.

En essensiell del av dette prosjektet var å holde klassene til sensoren og manageren organiserte og konsise, for å etterstrebe modularitet og effektivitet i kjernen av pakken vår. Både eksempelkoden vår og brukernes egen funksjonalitet blir bygget oppå denne. En konsis klasse vil nødvendigvis gå raskere å skaffe oversikt over, og bli lettere å lære og bygge videre på som bruker; altså er den brukervennlig. Det er også viktig å holde disse klassene effektive, og ikke bruke unødvendig med ressurser. Det har som hensikt å fri opp ressurser til andre formål.

Manager-klassen implementerer all MQTT-kommunikasjonen; både MQTT-serveren og klienten eksisterer i dette skriptet. Originalt var både klient og server egne filer utenfor manageren, men de var en innpakning til koden fra MQTTnet som tilsynelatende ga lite nytteverdi. Siden å dedikere egne filer til noen få linjer med kode, innlemmet vi både klient og server i manager-skriptet. I ettertid ser vi at det hadde lønnet seg å flytte klienten ut av dette skriptet, og inn i sitt eget. Hensikten med det hadde vært å gjøre koden enda mer universell, modulær og lettere å tilpasse til egne behov. For eksempel kunne en bruker bygget en egen klient som benytter en annen IoT-enhet, men alikevel tatt i bruk vår manager som server. Dette skriptet kunne vært inkludert i manager prefaben, og dermed ikke hatt noen effekt på brukervennligheten.

Et problem som oppstod under utviklingen av sensor-klassen var måten aktiv/inaktiv-status skulle håndteres. Sensoren sender bare pakker dersom et objekt oppdages, derfor var det nødvendig å implementere en måte å følge med på og oppdatere statusen til sensoren. Når det har gått en viss tid siden forrige pakke ble mottatt, går sensoren fra aktiv til inaktiv status og tømmer listen med oppdagede objekter. Dette intervallet

kan justeres av bruker. Dette er ikke en ideell løsning da det forårsaker en del treghet i overgangen fra aktiv til inaktiv. Dette er for eksempel uønsket i spillsammenheng. Dersom sensoren kunne sendt pakker for å indikere at den ikke oppdager noe, kunne oppdateringen blitt utført raskere. Dette er et symptom på at spillutvikling er utenfor sensoren sitt tiltenkte bruksområde, og kan også sees i oppdateringsfrekvensen til sensoren; pakker sendes bare hvert 100 millisekund. Dette er en betydelig forsinkelse i spill, spesielt de som er beregnet på å konkurrere med andre.

Siden sensoren sender pakker såpass sjeldent, ble det enda viktigere å bruke events til å utlyse at ny data er tilgjengelig. Selv om bruker kan sjekke listen hver bildeoppdatering, vil sensoren bare oppdatere denne hvert 100 millisekund eller sjeldnere. Events lar bruker oppdatere objektene bare når ny informasjon er kommet fra sensoren, som sparer 50 metodekall per sekund i et standard 60 bilder i sekundet spill. Selv om disse metodene i seg selv ikke er krevende vil unødvendige kall bygge opp systemkravene etterhvert som spill og prosjekter blir større. En ulempe med implementasjonen av events, er at koden straks blir litt mer kompleks og vanskelig å forstå.

Det tunge fokuset på brukervennlighet i utvikingen av pakken bidro til å senke tempoet på feature-utvikling. Spesielt da mye tid ble brukt på brukertester og administrativ belastning assosiert med disse. Det er imidlertid bedre å ha et tidlig fokus på brukervennlighet da dette sparer tid på lang sikt. Det er mer effektivt å gjøre et program brukervennlig fra starten enn å endre hele programmet etter masse funksjonalitet er ferdig utviklet.

5.1.1 Diskusjon av produktets funksjonelle krav

De funksjonelle kravene har gått igjennom et par iterasjoner; de startet veldig enkle i kravspesifikasjonen fra oppdragsgiver, de ble videreutviklet av gruppen, og til slutt har de blitt revidert av gruppen underveis i prosjektet siden vi har lært med om Unity og hva som kreves av pakken.

De aller fleste funksjonelle kravene har allikevel blitt oppnådd, bare 2 av 11 ble ikke implementert.

Det ene kravet som ikke ble implementert var funksjonalitet for å velge hvilken informasjon pakken skal motta fra en gitt sensor. Dette kravet ble forkastet fordi dette er mer hensiktsmessig å implementere på brukernivå. Alt innholdet i pakken fra sensoren blir allerede deserialisert, slik at det ikke vil spare noen ressurser fra vår side om vi forkaster

dataen. Tvert imot vil ekstra kode for å støtte denne funksjonaliteten øke ressursbruken marginalt. Vi lar heller bruker motta hele objekter, og benytte den informasjonen som er ønsket. Det vil ikke være mer ressusrbruk fra datamaskinens side, siden pakken alt er deserialisert.

Den andre funksjonen som ikke ble implementert er sonefunksjonaliteten til sensoren. Denne ble sett på som lite hensiktsmessig å implementere, da hovedvekten av nytteverdi kommer fra den "vanlige" funksjonaliteten. Det er mulig for bruker å implementere sin egen sonefunksjonalitet i Unity om dette er ønskelig.

5.1.2 Diskusjon av produktets ikke-funksjonelle krav

5.1.2.1 Dokumentasjon

Dette var spesielt viktig da både de som tar i bruk pakken for funksjon og de som skal videreutvikle kommer til å lese dette. Mangel på dokumentasjon er en av de vanligste brukervennlighetsproblemene med kodebiblioteker. Gruppen var klar fra starten av prosjektet at koden skulle være fult dokumentert, og dokumentering var del av kodegjennomgangen i prosjektet. Dette gjorde det lett å oppnå målet, men å bruke mye tid på å skriving og oppdatering av sekundærnotasjon som dokstrenger gjennom prosjektet er også veldig tidkrevende. Det kan være ugunstig å bruke tid på dokumentasjon av en metode som senere viser seg å være unyttig og dermed blir fjernet fra kodebasen.

5.1.2.2 Brukergrensesnitt

Kravet om et brukersnitt som er "klart, tydelig og enkelt å forstå", gjorde det nødvendig å benytte og utvide Unity sitt grensesnitt i to tilfeller. Unity tilrettelegger for modifikasjon av brukergrensesnitt, som gjorde endringene enkle å implementere. Implementasjonen av "Add Sensor"-knappen er en slik utvidelse av Unity sitt brukergrensesnitt, og utseendet vil dermed samsvare med innstillingene satt i Unity og alltid følge universell utforming. Knappen fikk gode tilbakemeldinger fra brukertestene. En fordel med å utvide Unity sitt brukergrensesnitt og å følge standardene for grensesnitt, er at brukere med erfaring fra Unity vil klare å ta i bruk grensesnittet enklere.

5.1.2.3 Feilsøking

Under utvikling ble loggføring av alle meldinger fra klient, server og sensor fort overveldende. Vi tok inspirasjon fra MQTTnet sin logger, og implementerte et eget system for nivåstyring oppå Unity sin konsoll. Dette gjør at vi kan forkaste unyttige meldinger, og få bedre oversikt over hva som er galt.

En ulempe her er at vi bruker flere ressurser, da vi både må instansiere en logger fra MQTTnet, og en wrapper for denne som videresender meldinger til Unity. For å motvirke dette har vi gjort det mulig å slå av logging helt, som kan være nyttig i ferdige prosjekter. Vi har også sørget for at logging blir automatisk avslått i kompilerte prosjekter for å unngå unødvendig ressursbruk.

5.1.2.4 Brukseksempel

Vi hadde mål

Eksempelkoden vår har blitt utviklet for å vise brukere hvordan man kan ta i bruk pakken vår. Fordelen med denne koden er at brukere raskere kan komme i gang med pakken vår. Det kan være at koden er implementert på en ueffektiv eller vanskelig måte, og da vil i så fall eksempelkoden være støy i læringsprosessen.

5.1.2.5 Utvidet funksjonalitet

Mye av fokuset var rundt å gjøre kjernen til programmet manager- og sensor-klassene leselige og redusere ressursbruken til disse. Dette var fordi disse klassene ikke skal endres av brukeren, bare kodes opp mot og utvides av brukeren. Dette betydde at disse klassene ikke burde ha unødvendige fuksjoner som brukeren ikke ønsker å bruke resurser på. Det var for eksempel derfor logging av 5-minutters-meldingene sensoren sender ble laget som en egen klasse som kunne legges til ved behov.

5.1.2.6 Universell utforming

Mye av brukervennligheten og nesten alt tilgjengeligheten til pakken kommer fra Unity sitt eget grensesnitt som er utenfor gruppens kontroll. Hvis en funksjon i Unity var vanskelig

å finne kunne gruppen prøve å forklare denne i README-filen til prosjektet, men ikke endre dette. Tilgjenglighets-funksjoner som større tekst og lignende er også innebygd i Unity, så det var ikke nødvendig for gruppen å forholde seg til dette.

5.1.2.7 Installasjon

Vi distribuerer dll-filen til MQTTnet sammen med pakken vår. Dette fører til en svært enkel og brukervennlig installasjonsprosess for pakken vår, men er bare mulig på grunn av den ettergivende MIT-lisensen pakken utgis med. Den tillater fri redistribusjon av programvaren, men kan ha bieffekten av at den skjuler annerkjennelsen til den originale produsenten. For å ta hensyn til dette har vi skrevet at vi bruker MQTTnet på toppen av README-en vår.

Siden avhengigheten til pakken distribueres sammen med pakken i versjonskontrollen, vil den ikke oppdatere seg automatisk. Dette kan få negative konsekvenser dersom den ikke blir oppdatert til versjoner som tetter eventuelle sikkerhetshull. På den andre siden vil avhengigheten være veldig stabil.

5.1.2.8 Pålitelighet

Også her er vi både begrenset og hjulpet av at vi er låst til Unity. Unity er en veletablert spillmotor som er godt anerkjent i spillutviklingsmiljøet, og vi kan dermed regne med at denne vil kjøre svært pålitelig. Sensorens stabilitet er også utenfor vår kontroll, men da sensoren er laget for å overvåke bygg og uteområder over lengre tid er den trolig relativt driftsikker.

5.1.2.9 Testdekning

Det er derimot viktig at vi tester koden vår godt, slik at pakken er stabil i bruk. Vi har derfor etterstrebet høy testdekningsgrad, og oppnådde til slutt 60%.

Det hadde vært gunstig å sette av mer tid til å lære mer om spillemodus-tester sette opp en "falsk klient" som sender data i samme format som sensoren. Denne klienten kunne blitt benyttet i testing og til å utvikle opp mot "sensoren" uten å ha en fysisk sensor tilgjengelig. Den kunne i tillegg latt brukere teste pakken i prosjektet sitt uten en fysisk

sensor, dersom de var usikre på om de ville anskaffe sensoren til sitt prosjekt.

5.1.2.10 Kryssplattformsstøtte

Det var veldig lett å gjøre pakken kryssplattform, siden Unity er designet for å gjøre det lett for brukere å lage kryssplatformsprosjekter. Dette kunne blitt et større problem dersom pakkens MQTT-håndtering hadde vist seg å ha behov for funksjoner som er utenfor det som er inkludert i .NET Standard 2.1.

5.1.2.11 Langsiktig vedlikehold

Siden pakken trolig skal brukes av Ablemagic og andre over lengre tid, var det viktig med langsiktig vedlikehold. Vedlikeholdet vil mest sannsynlig ikke bli utført av gruppemedlemene, derfor ble pakken laget på en måte som gjorde det lett for andre å vedlikeholde. Dette ble oppnådd ved å fokusere på ren kode, brukervennlighet, og å gjøre koden så leselig som mulig. God dokumentasjon vil også bidra til dette. Dersom en bruker oppdager at en metode ikke lenger fungerer som den skal kan brukeren benytte dokumentasjonen til å forstå hva metodens intensjon var. Testing bidrar også til dette ved å gi de som eventuelt ønsker å endre koden en indikasjon på eventuelle feil som følger med endringen de ønsker å innføre. Videre har vi valgt stabil, annerkjent programvare å bygge pakken vår på.

Grunnen til å gjøre koden til åpen kildekode var blant annet at åpen kildekode oppmuntrer til samarbeid om utvikling og vedlikehold av prosjekter [23]. Dersom alle brukere har fult innsyn i koden vil en bruker som ønsker en utvidet funksjon kunne lage denne og gjøre den tilgjengelig for alle andre brukere.

5.1.2.12 Sikkerhet

En svakhet med systemet vårt er at det ikke er kryptert kommunikasjon mellom den fysiske sensoren og MQTT-servern i manageren vår. Resten av pakkeflyten burde i teorien være trygg, da de går over local loopback, og ikke wifi. Dersom det er en inntrenger som har tilgang til local loopback vil dette også være utrygt.

Under etablering av kommunikasjon vil brukerkonfigurert passord og brukernavn sendes

i åpen tekst over wifi, siden kryptert kommunikasjon ikke ble implementert. Dette er et klart brudd på målet vårt om å ikke skape flere sikkerhetshull. Vi har implementert muligheten til å skrive inn brukernavn og passord i Unity sitt brukergrensesnitt, og en bruker kan anta at disse da er trygge å fylle ut med sensitiv informasjon eller passord som blir brukt andre steder. Det blir informert om problemet i README-filen, men en bedre løsning hadde vært å skjule disse feltene inntil problemet er løst. Et annet mindre problem ved at brukernavn og passord blir sendt åpent, er at en ondsinnet aktør kan koble opp mot MQTT-serveren og utgi seg for å være sensoren.

Det ble gjort et forsøk på å implementere kryptert kommunikasjon, men dette måtte etterhvert nedprioriteres da det tok for lang tid å feilsøke. Det var tungvint å feilsøke problemet, da Unity helst ser at du bruker de ferdigimplementerte metodene for kommunikasjon. Disse metodene er beregnet for kommunikasjon mellom forskjellige Unity-instanser, både servere og klienter, og ikke utenomstående programvare. Vi forsøkte å lage vår egen sertifikatsverifikasjonskjede som godtar alle sertifikater, og avsluttet forsøket når vi fortsatt fikk beskjed om at sertifikatet ikke ble stolt på av Unity.

5.2 Diskusjon av vitenskaplige resultater

5.2.1 Første runde brukertester

Som beskrevet over hadde gruppen en hypotese om at å fokusere på brukervennlighet tidlig i prosjektet ville redusere mengen brukervennlighetsproblemer oppdaget i brukertester. Mye av valgene beskrevet over ble gjort basert på egen erfaring fra arbeid med lite brukervennlige programvarebibliotek, men gruppen ønsket også å ha en formell forståelse for brukervennlighet å underbygge dette med. Første runde med brukertester oppdaget etter gruppens vurdering få problemmer. Dette er trolig på grunn av gruppens fokus på å gjøre pakken brukervennlig fra starten av prosjektet.

Noen problemer med brukertest metodikken ble oppdaget i første runde med brukertester. Testbrukerene følte at de ble satt på prøve og at vi vurderte deres kodeevne. Dette førte til en høy grad av stress i testpersonen og påvirket tilsynelatende deres vilje til å ta seg tid til å sette seg inn i koden før de prøvde å løse oppgaver. Det ble foreslått å gjøre la brukeren gjennomføre testen på egenhånd med skjemopptak som gruppemedlemmene kunne se gjennom etter testen. Dette ble ikke gjort da den fysiske sensoren måtte være tilstede under testene og gruppen vurderte løsningen beskrevet over som tilstekkelig. På

grunn av behovet for den fysiske sensoren og WiFi som sensoren kunne benytte var det allerede store krav til lokasjon for tester. Å sette opp testene så bruker kunne være alene var for krevende på grunn av dette. Det var også upassende for gruppa å gi sensoren til en bruker uten tilsyn da sensoren tilhører Ablemagic. Det ble bestemt å åpne med en beskjed om at det var pakken som var på prøve ikke bruker i andre omgang med brukertester.

Spørsmål om erfaringsnivå gjorde mindre erfarene testpersoner usikre på seg selv. Målet med å ha spørsmål om erfaringsnivå var å få en ide om brukeren var en nybegynner eller ikke slik at gruppen kunne se om noen problemer gikk igjen hos nybegynner eller om de var universelle. Dette ble oppnådd da gruppen fant noen problemer som gikk igjen hos de med mer erfaring med Unity og fikk adressert disse, men den negative effekten på testpersoner som antok de var mindre erfarene enn gruppemedlemmene var et stort problem.

I første oppgave hadde mange problemer med SSH. Dette var på grunn av at GitHub hadde endret Secure Socket Shell (SSH) vertsnøkkel dagen før [38]. Gruppen oppdaget dette problemet under en av testene og oppfordret i senere brukere å bruke Hypertext Transfer Protocol Secure (HTTPS). Dette påvirket de første tre testene i denne runden negativt da disse brukerene ikke fikk installert pakken som forventet.

I samme oppgave var det en av brukerne som etterspurte en annen installasjonsmåte. Dette ble vurdert og planlagt gjennomføring av, men ble ikke prioritert i forhold til andre oppgaver. Denne installasjons metoden er også mindre vanlig og en eldre versjon av pakke installering som Unity mulig kommer til å fjerne i fremtiden. Hallgeir Løkken ga tilbakemelding til gruppen om at denne funksjonen ikke var ønsket av Ablemagic, dette var del av grunnen til at gruppen ikke prioriterte dette.

Under testene ble det klart at måten sensorobjektet var implementert ikke var intuitivt. Det ble bestemt å lage en knapp som la til sensorer på manager for brukeren. Dette ble valg over å legge til sensorer automatisk dersom de ikke allerede fantes nekter brukeren kontroll over hvilke sensorer de vil håndtere. Det er mulig at en bruker har to sensorer på samme nett, men bare ønsker å håndtere pakker fra en av dem. Å legge til sensorer automatisk forekler prosessen for brukeren, men fjerner kontroll. Gruppen prioriterte å gi brukeren kontroll over hvilke sensorer de ønsker å håndtere.

Mange av brukerene hadde problemer med å finne eksempelkoden i mappestrukturen og alle brukte ekstra tid på dette. Det var ikke intuitivt å ha den inne i runtime-mappa. Det ble bestemt å flytte eksempelkoden til rotmappen så den ble mer synlig for brukeren.

Dette er også i tråd med Unity sine beste praksiser i forhold til pakke struktur. Unity foreslår imidlertid å gjøre denne pakken usynlig i Unity sitt grensesnitt noe gruppen ikke valgte å gjøre da Unity sitt grensesnitt gjennom testen var brukeres primære måte å finne skript på.

5.2.2 Andre runde brukertester

Beskjeden om at vi ikke vurderte brukeres egenskaper hadde en stor positiv effekt på brukeres stress under andre runde med brukertester. Brukere under denne testen var mer komfortable og mer villige til å ta seg tid til å sette seg inn i koden. Denne beskjeden ga også i praksis brukeren informasjon om at å bli nervøs er vanlig og dermed ble dette normalisert for bruker.

Problemet med at å oppgi erfaringsnivå gjorde brukere usikre ble klart utbedret i andre omgang med brukertester. Endringen til å spørre brukere om å gi en tallverdig gjorde trolig at brukerene ikke følte at de måtte forsvare engenskapene sine. Det var tydelig lettere for en bruker å si at de var for eksempel 4 i erfaringsnivå enn at de var over middels erfarende. Selvom disse påstandene er tillsvarende var brukere mer komfortable med tallverdier.

Knappen i Unitys grafiske grensesnitt var vanskelig å se. Dette er et problem med Unity sitt grensesnitt. Vi kunne endret knappen sitt utseende til å være mer synlig, men dette ville brutt med Unity sin universelle utforming. Hvis brukeren har innstillinger som øker synlighet, vil en standard knapp følge disse. Om vi hadde justert knappen til en annen farge ville den forblitt denne fargen. Dette gjelder selv når en bruker som har konfigurert Unity til å være den samme fargen som for eksempel bakgrunnen.

Det er mulig at å beholde en prefab for sensor som en alternativ måte å legge til på hadde vært nyttig, da dette hadde latt brukerene velge den metoden de fant lettest. De fleste synes fortsatt knappen var en god løsning som gjorde det lettere å legge til sensorer.

Det ble ikke gjort klart for brukere hvilken filtype filen med brukernavn og passord skulle være. Dette var et problem for mange av brukerene, selvom gruppen la merke til at brukere med erfaring fra arbeidslivet hadde mindre problemer. Dette var trolig på grunn av at de har tidligere erfaring med hemmelige filer som må lagres bare lokalt. Det ble tydelig at dette måtte klargjøres i dokumentasjonen. Det var også et problem for mange brukere at å lage en tekstfil krevde at de navigerte ut av Unity sitt grensesnitt da Unity ikke lar

brukere opprette tekstfiler. Det ble vurdert å gjøre om til en metode hvor brukernavn og passord var variabler, men dette ville gjort det umulig å holde disse hemmelig hvis prosjektet ble delt på nett. Det ble derfor bestemt å utbedre dokumentasjonen.

Problemet med at brukere ikke fikk informasjon om filer ble generert automatisk var lignende til det over. Brukere hadde problemer med å lage filer av en type som ikke kunne lages i Unity sitt brukergrensesnitt. Det er dermed lite gunstig å få brukere til å gjøre dette unødvendig. Målet med å la brukeren oppgi en path til lagringsfilen var å gi brukeren muligheten til å plassere filen der de ønsket selv. Dette er en nødvendig funksjon da brukere har potensielt store spillprosjekter med en bestemt filstruktur og pakken vår burde ikke motstille seg denne. Det vil imidlertid være brukere som ikke har en preferanse til hvor filen skal lagres som bare vil enkelt sette opp logging. Det var implementert en løsning som dekker begge disse behovene, men det var ikke klart for bruker at den enklere metoden fantes.

Det ble klart under brukertestene at bruken av UnityEvents ikke var godt nok forklart. Brukere lette etter dette i dokumentasjonen og i koden, men fant det de lette etter nesten utelukkende ved å få øye på riktig variabelnavn. Selv om det var dokumentert med kodedokumentsjon var det ikke forklart på en måte som brukere forsto lett. Det var ikke dokumentert godt nok i ekstern dokumentasjon som README.

Under den siste brukertesten med Rinsai ble det oppdaget mange problemer som ikke hadde oppstått med andre brukere i hverken denne eller forrige runde med tester. Dette var fordi Rinsai hadde mye mer erfaring som Unity-utvikler enn noen annen testperson. Det ble etterspurt mye mer dokumentasjon i Unity sitt grensesnitt. Dette var gruppen tidligere ikke kjent med og hadde derfor ikke implementert dette. På grunn av at Ablemagic hadde et viktig prosjekt pågående ble denne testen gjennomført veldig sent i prosjektsperioden. Å få informasjon om denne måten å ha dokumentasjon på tidligere i prosjektet hadde vært veldig nyttig og ville latt gruppen utvikle et enda bedre brukergrensesnitt for pakken. Det er ukjent om noen andre brukere savnet denne typen dokumentasjon uten å uttrykke dette. Hvis ikke andre brukere av Unity kjenner til denne dokumentasjonen, er dette et mindre problem. Om dette bare er et problem for svært erfarne Unity-utviklere, er disse brukerne også de brukerne som trenger minst tilrettelegning for å benytte pakken.

5.3 Diskusjon av administrative resultater

5.3.1 Fremdriftsplan

Det ble gjort store endringer til planen etter forprosjektsplanen ble laget. Dette var fordi problemstillingen ble endret så den originale planen om å holde demonstrasjon av sensoren ikke lenger tjente utforskning av problemstillingen. Denne endringen ble gjort svært tidlig i prosjektet da gruppen ønsket å best mulig tjene målsetningene i prosjektet.

Det ble også opprinnelig bestemt å ikke endre koden etter siste brukertest. Dette ble gått bort i fra da brukertestens oppdagede problemer var lette nok å fikse til at gruppen kunne avsette tid til å forbedre produktet. Pakken sin dokumentasjon ble mye bedre som resultat av denne endringen i planen. Det er trolig at endringen i planen førte til forbedringer i README-filen som vil gjøre pakken mer innbydende for brukere og dermed mer sannsynlig til å bli brukt.

5.3.2 Arbeidsprosessen

Det ble naturlig for prosjektet å ha ikke ha en alt for streng arbeidsflyt, med tanke på at gruppen bare besto av to personer og arbeidet ble gjennomført i fysisk nærvær. Kommunikasjon var derfor ikke et stort problem. Å implementere en altfor rigid arbeidsflyt ville vært unødvendig tidkrevende, i forhold til hvor mye nytteverdi gruppen ville ha opplevd.

De strenge utviklingsrammene som ble satt for prosjektet, som kodegjennomgang, ble forsikret med regler på GitHub. Dette var på grunn av erfaring fra tidligere prosjekter gruppen hadde hatt sammen. I disse prosjektene hadde kodegjennomgang blitt droppet i perioder der det var lite tid før en frist, noe som introduserte problematisk kode oftere. Å håndheve strenge rammer for prosjektet programmatisk hadde positive og negative effekter. Det gjorde at gruppen holdt seg til rammene og gjorde utviklingen mer systematisk. Imidlertid opplevde gruppen utfordringer i perioder hvor et av medlemmene var utilgjengelig. Dette førte til at det andre medlemmet ikke kunne fullføre kodingen av prosjektene de hadde begynt på før det første medlemmet returnerte.

5.3.3 Samarbeid

Det har vært godt samarbeid innad i gruppen under prosjektperioden. Tidlig i prosjektet ble en felles enighet om mål, verdier og ambisjoner for arbeidet etablert. Disse har vært gode for å bruke som retningslinjer for å få en enighet om hvordan prosjektet skulle gjennomføres. God planlegging og jevnt arbeid har vært essensielt for å nå de felles målene vi satte oss i starten av prosjektet. God kommunikasjon og forståelse fra begge parter har ført til at mindre faglige uoverensstemmelser alltid har gått fint.

Ablemagic stilte med kontorplass og denne har bidratt til en godt samarbeid. Å ha tilgjengelig et arbeidslokale gjorde det lett å sette opp fast arbeidstid og møteplass i prosjektperioden. Dette reduserte administrativ belastning i samarbeidet da gruppen slapp å diskutere oppmøte tid og sted før hver arbeidsdag. Å sitte sammen fysisk gjorde også kommunikasjonen rask og effektiv og bidro til gjensidig hjelp og problemløsning. Ved å ha fast møteplass og oppmøtetidspunkt kunne besluttninger bli raskt diskutert og tatt. Dette gjorde forsinkelser i beslutninger små under arbeidet.

5.4 Samfunnspåvirking

Gruppen har forsøkt å gjøre pakken så ressurseffektiv som mulig og vil ikke i noen stor grad påvirke energikravene til brukeres prosjekter. Dette var viktig da ressursbruk i spill er en av de begrensende faktorene for hva som kan lages.

Vår kode vil sannsynligvis ikke ha en stor samfunnspåvirking, men kunstinstallasjoner som kan lages ved hjelp av koden kan ha stor påvirkning. Dette kan for eksempel skje igjennom dulting, små uformelle påvirkninger som er designet for å påvirke adferd i gunstige retninger. Integreringen mellom Unity og radarsensoren kan være egnet til å styre installasjoner som kan gjøre slike "dult". Kunst, by- design, utseendet og bymiljø påvirker folks liv. Integrering av levende installasjoner kan brukes til å gjøre områder mer innbydende og oppfordre til aktiv lek. Siden vår pakke kan brukes til å slå av og på installasjoner som bruker mye energi slik at de bare bruker strøm når noen er tilstede for å se installasjonen. Dette vil spare miljø konsekvensene ved for eksempel å redusere lysforurensning som kan negativt påvirke økosystemer [39] og kanskje spesielt insekter. Styring av lysinstallasjoner sparer penger og muliggjør at kommuner og andre aktører vil kunne lage få råd til å sette opp installasjoner som forbedrer bymiljø og øker trivsel til innbyggerene.

Det er vist at pene bygninger er utsatt for mindre hærverk [40] det er mulig at å sette

opp disse installasjonene kan bidra til å gjøre byen mer estetisk og som konsekvens senke mengden hærverk på bygg. Spesielt for bygg som ikke aktivt er i bruk. Siden hærverk er en selvforsterkende prosess [40] kan disse installasjonene bidra til å bryte denne prosessen. Siden pakken er aktiv implementert opp mot en radarsensor dette kan foreksempel muliggjøre å belyse folk eller dyr som krysser vegen på kjente trafikkerte overganger om muligens bidra til økt trafikksikkerhet.

6 Konklusjon og videre arbeid

6.1 Hvordan kan vi utvikle programvare som er brukbar for andre utviklere?

Denne oppgaven utforsket hvordan man utvikler programvare som er brukervennlig for andre utviklere. Problemstillingen ble utforsket ved å gjennomføre et litteraturstudie, og deretter ble dette satt ut i praksis ved å organisere, utføre og analysere to omganger med brukertesting.

Etter litteraturstudiet og brukertestene ble det identifisert mange elementer som inngår i å gjøre et programvarebibliotek brukervennlig, men det er to elementer som utmerket seg som spesielt viktige: synlighet og rolle-uttrykksevne. Synlighet kan utbedres ved hjelp av god navngiving og tydelig pakkestruktur. Rolle-uttrykksevne kan økes med dokumentasjon, også her er navngiving viktig.

Det var tydelig for gruppen at fokuset på brukervennlighet i første og andre sprint bidro til få brukervennlighetsproblemer i brukertesten som etterfulgte andre sprint. Det ble oppdaget få problemer og bare ett problem av høy alvorlighetsgrad. Gjennom prosessen ble problemer med testmetodikken oppdaget og effektivt utbedret. Dette førte til at andre brukertest var langt mer effektivt gjennomført. Det endelige produktet er en brukervennlig, robust og effektiv pakke til Unity spillmotoren. Den er utviklet iterativt og veltestet.

Det er enda mye rom for å utforske brukervennlighet i utvikling. Siden de fleste studier av brukervennlighet for API-er er brukertester, er det et behov for mer kvantitativ forsking i dette feltet.

6.2 Videreutvikling

Det viktigste potensielle området for videreutvikling, er kryptert kommunikasjon mellom sensoren og maskinen som kjører Unity-prosjektet. Kommunikasjonen foregår over wifi, og en potensiell uønsket tredjepart kan enkelt lese informasjon i pakker via Wireshark eller lignende programvare. Informasjonen i pakkene er ikke av spesielt sensitiv natur, da sensoren i seg selv er personvernsvennlig, men det kan allikevel være hensiktsmessig

å implementere kryptert kommunikasjon da sensoren sender brukernavn og passord i ren tekst ved etablering av koblingen. Med brukernavnet og passordet får en ondsinnet aktør forfalsket en sensor. Dette har kanskje ikke store konsekvenser i spill og kunstprosjekter, men dersom Unity har sikkerhetshull kan angriperen eskalere angrepet.

Testingen har også rom for forbedring; man kan forfalske sensordata via en egen MQTT klient, og utnytte dette til å teste flere funksjoner i pakken. Et annet bruksområde for forfalskning av sensordata er å la folk implementere pakken i sitt Unity prosjekt, og teste eller utvikle mot den uten å ha tilgang til en fysisk sensor.

Et potensielt område for videreutvikling av pakken, er å integrere sonefunksjonaliteten til sensoren. Sensoren tilbyr funksjonalitet for opp til 5 brukerdefinerte inngangporter og soner [3], men under prosjektet ble ikke denne funksjonaliteten utforsket. Portene kan gi beskjed når folk krysser, og i hvilken retning. Soner kan holde tellingen på antall mennesker i sonen i tillegg til mennesker som går inn og ut av sonen. En mulig måte å benytte soner og porter i pakken ville vært å implementere en UnityEvent for meldinger fra porter og soner.

En annen funksjon fra sensoren som ikke ble implementert i pakken vår, var muligheten til å styre innstillingene på sensoren via MQTT. For å justere hvilken type pakker sensoren sender, lang eller kort versjon, var man nødt til å koble seg opp mot webgrensesnittet til sensoren. En utbedring på dette området vil være muligheten til å justere sensoren sine innstillinger via Unity sitt grensesnitt over MQTT.

Referanser

[1] Irum Rauf, Elena Troubitsyna, and Ivan Porres. A systematic mapping study of API usability evaluation methods. *Computer Science Review*, 33:49–68, August 2019. ISSN 1574-0137. doi: 10.1016/j.cosrev.2019.05.001. URL https://www.sciencedirect.com/science/article/pii/S1574013718301515.

[2] Droidsonroids. What is Scrumban? | Definition and 5 Common Scrumban Myths, September 2021. URL https://www.thedroidsonroids.com/blog/what-is-scrumban-definition-and-common-myths. Section: Blog.

[3] SensMax. Outdoor people counting radar sensor SensMax TAC-B 3D-WP, 2018. URL https://sensmax.eu/devices/sensmax-tac-b-3d-w-radar-people-counter/.

[4] A. Blackwell, C. Britton, Anna Cox, Thomas Green, C.A. Gurr, Gada Kadoda, Maria Kutar, Martin Loomes, Chrystopher Nehaniv, Marian Petre, C.R. Roast, Chris Roe, A. Wong, and R. Young. Cognitive Dimensions of Notations: Design Tools for Cognitive Technology. In *Cognitive Technology 2001*, pages 325–341. Springer Berlin / Heidelberg, January 2001. ISBN 978-3-540-42406-2. doi: 10.1007/3-540-44617-6_31. Journal Abbreviation: Cognitive Technology 2001.

[5] Richard Coppen. OASIS MQTT Technical Committee Minutes of for the meeting of Thursday, 25th April 2013 Teleconference, April 2013.

[6] Felipe Pezoa, Juan L. Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. Foundations of JSON Schema. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, pages 263–273, Republic and Canton of Geneva, CHE, April 2016. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-4143-1. doi: 10.1145/2872427.2883029. URL https://dl.acm.org/doi/10.1145/2872427.2883029.

[7] Teknologirådet. Hva er tingenes internett?, January 2015. URL https://teknologiradet.no/hva-er-tingenes-internett/.

[8] Chris M. Lonvick and Tatu Ylonen. The Secure Shell (SSH) Protocol Architecture. Request for Comments RFC 4251, Internet Engineering Task Force, January 2006. URL https://datatracker.ietf.org/doc/rfc4251. Num Pages: 30.

[9] Roy T. Fielding, Mark Nottingham, and Julian Reschke. HTTP Semantics. Request for Comments RFC 9110, Internet Engineering Task Force, June 2022. URL https://datatracker.ietf.org/doc/rfc9110. Num Pages: 194.

[10] Eric Rescorla. HTTP Over TLS. Request for Comments RFC 2818, Internet Engineering Task Force, May 2000. URL https://datatracker.ietf.org/doc/rfc2818. Num Pages: 7.

[11] Martin Reddy. *API Design for C++*. Elsevier, March 2011. ISBN 978-0-12-385004-1. Google-Books-ID: IY29LylT85wC.

[12] Tilsynet for universell utforming av IKT. WCAG sortert etter prinsipp. URL https://www.uutilsynet.no/wcag-standarden/wcag-sortert-etter-prinsipp/713.

[13] Andrew Banks, Ed Briggs, Ken Borgendale, and Rahul Gupta. MQTT Version 5.0, March 2019. URL https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html.

[14] The HiveMQ Team. Publish & Subscribe - MQTT Essentials: Part 2, January 2015. URL https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe/.

[15] Charlie Wang. HTTP vs. MQTT: A tale of two IoT protocols, November 2018. URL https://cloud.google.com/blog/products/iot-devices/http-vs-mqtt-a-tale-of-two-iot-protocols.

[16] Amy Moormann Zaremski and Jeannette M. Wing. Signature matching: a tool for using software libraries. *ACM Transactions on Software Engineering and Methodology*, 4(2):146–170, April 1995. ISSN 1049-331X. doi: 10.1145/210134.210179. URL https://dl.acm.org/doi/10.1145/210134.210179.

[17] Red Hat. What is an API?, February 2022. URL https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces.

[18] Rafael Valencia-García, Katty Lagos-Ortiz, Gema Alcaraz-Mármol, Javier del Cioppo, and Nestor Vera-Lucio. *Technologies and Innovation: Second International Conference, CITI 2016, Guayaquil, Ecuador, November 23-25, 2016, Proceedings*. Springer International Publishing, October 2016. ISBN 978-3-319-48023-7. Google-Books-ID: 1vcMvgAACAAJ.

[19] TrustRadius. List of Top Game Engine Software 2023, 2023. URL https://www.trustradius.com/game-engine.

[20] Brian Fitzgerald and Klaas-Jan Stol. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, 123:176–189, January 2017. ISSN 0164-1212. doi: 10.1016/j.jss.2015.06.063. URL https://www.sciencedirect.com/science/article/pii/S0164121215001430.

[21] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access*, 5:3909–3943, 2017. ISSN 2169-3536. doi: 10.1109/ACCESS.2017.2685629. Conference Name: IEEE Access.

[22] Eric S. Raymond. *The cathedral and the bazaar : musings on Linux and Open Source by an accidental revolutionary*. Beijing ; Cambridge, Mass. : O'Reilly, 2001. ISBN 978-0-596-00108-7 978-0-596-00131-5. URL http://archive.org/details/cathedralbaz00raym.

[23] Open Source Initiative. The Open Source Definition, July 2006. URL https://opensource.org/osd/.

[24] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Pearson, Upper Saddle River, NJ, 1st edition edition, August 2008. ISBN 978-0-13-235088-4.

[25] Haley Hunter-Zinck, Alexandre Fioravante de Siqueira, Váleri N. Vásquez, Richard Barnes, and Ciera C. Martinez. Ten simple rules on writing clean and reliable open-source scientific software. *PLOS Computational Biology*, 17(11):e1009481, November 2021. ISSN 1553-7358. doi: 10.1371/journal.pcbi.1009481. URL https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1009481. Publisher: Public Library of Science.

[26] Heather L. Keenan, Simon L. Duke, Heather J. Wharrad, Gillian A. Doody, and Rakesh S. Patel. Usability: An introduction to and literature review of usability testing for educational resources in radiation oncology. *Technical Innovations & Patient Support in Radiation Oncology*, 24:67–72, December 2022. ISSN 2405-6324. doi: 10.1016/j.tipsro.2022.09.001. URL https://www.sciencedirect.com/science/article/pii/S2405632422000324.

[27] ISO. ISO 9241-11:2018(en), Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts, 2018. URL https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-2:v1:en.

[28] Carol M. Barnum. 1 - Establishing the essentials. In Carol M. Barnum, editor, *Usability Testing Essentials*, pages 9–23. Morgan Kaufmann, Boston, January 2011. ISBN 978-0-12-375092-1. doi: 10.1016/B978-0-12-375092-1.00001-5. URL https://www.sciencedirect.com/science/article/pii/B9780123750921000015.

[29] Y.K. Dwivedi, N.P. Rana, A. Jeyaraj, M. Clement, and M.D. Williams. Re-examining the Unified Theory of Acceptance and Use of Technology (UTAUT): Towards a Revised Theoretical Model. *Information Systems Frontiers*, 21(3): 719–734, 2019. ISSN 1387-3326. doi: 10.1007/s10796-017-9774-y.

[30] Jakob Nielsen. Usability 101: Introduction to Usability, January 2012. URL https://www.nngroup.com/articles/usability-101-introduction-to-usability/.

[31] Rahul Kamal Bhaskar, Craig Anslow, John Brosz, and Frank Maurer. Developing usable APIs with XP and cognitive dimensions. In *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 101–105, September 2016. doi: 10.1109/VLHCC.2016.7739671. ISSN: 1943-6106.

[32] T. R. G. Green and M. Petre. Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework. *Journal of Visual Languages & Computing*, 7(2):131–174, June 1996. ISSN 1045-926X. doi: 10.1006/jvlc.1996.0009. URL https://www.sciencedirect.com/science/article/pii/S1045926X96900099.

[33] Andrew Sears and Julie A. Jacko. *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications, Second Edition*. CRC Press, September 2007. ISBN 978-1-4106-1586-2. Google-Books-ID: A8TPF_O385AC.

[34] Max Rehkopf. What is a Kanban Board?, 2023. URL https://www.atlassian.com/agile/kanban/boards.

[35] ProductPlan. Scrumban, 2023. URL https://www.productplan.com/glossary/scrumban/.

[36] Dan Radigan. Standups for agile teams, 2023. URL https://www.atlassian.com/agile/scrum/standups.

[37] Tashia T. 15 Most Popular GitHub Repositories Every Developer Should Know, March 2023. URL https://www.hostinger.com/tutorials/most-popular-github-repos.

[38] Mike Hanley. We updated our RSA SSH host key, March 2023. URL https://github.blog/2023-03-23-we-updated-our-rsa-ssh-host-key/.

[39] Arne Follestad. *Effekter av kunstig nattbelysning på naturmangfoldet - en litteraturstudie*. Norsk institutt for naturforskning, 2014. ISBN 978-82-426-2700-1. URL https://brage.nina.no/nina-xmlui/handle/11250/2388109. Accepted: 2015-03-02T08:40:43Z ISSN: 1504-3312 Publication Title: 89 s.

[40] Arnold P. Goldstein. *The Psychology of Vandalism*. Springer Science & Business Media, June 2013. ISBN 978-1-4899-0176-7. Google-Books-ID: ByfLBgAAQBAJ.

A Kravspesifikasjon fra ablemagic

Fra: Hallgeir Løkken
Sendt: mandag 16. januar 2023 kl. 08:03
Til: Oda Alida Fønstelien Hjelljord; Erik Borgeteien Hansen; Elise Klæbo Vonstad
Emne: Bacheloroppgave acceptance test

Hei!
Her er hvordan jeg ser for meg en acceptance test kan være.

«Studentene presenterer resultatet ved prosjektets avslutning. Presentasjonen skal holdes i Ablemagic sitt studio. I presentasjonen skal en SensMax Tac-B sensor være satt opp og kommunisere med en av studentenes PC hvor Unity kjører. Studentene skal vise til Ablemagic hvilken data som mottas fra radaren, og hvordan den presenteres i spillmotoren.
Som et minimum skal dataen som mottas vise todimensjonal posisjon av mennesker i radarens deteksjonsfelt.
I rapporten som leveres etter presentasjonen skal det være en kort drøfting av radarens begrensninger, med eksempel på situasjoner den sliter med deteksjon, eller situasjoner den produserer falske positiv.»

Det er holdt ganske enkelt, men om dere får til noe som det beskrevet over er jeg fornøyd. Veldig bra om dere også kan ta med estimert hastighet per menneske. Eksempler på situasjoner jeg mener dere bør teste for rapporten jeg ønsker meg til slutt er barn/voksne, ting som beveger seg i deteksjonsfeltet som ikke er mennesker, rullestolbrukere/mennesker som går med krykker, og mennesker som går tett. Dette er jo opp til radaren hva den klarer å se, så jeg mener ikke at dere skal gjøre noe for å forbedre dette, bare finne ut av hva akkurat denne radaren er god på/dårlig på, siden dere kommer til å bruke en god del tid med den. Dette vil hjelpe oss bestemme når vi vil bruke den ute på en installasjon eller ikke.

Hallgeir Løkken
Lead Developer
Skippergata 11, 7042 Trondheim
hallgeir@ablemagic.no
www.ablemagic.no

B Forprosjektsplan

Prosjekt 22

1 Mål og rammer

1.1 Orientering

Denne oppgaven er fremlagt av Able Magic AS til studentene, og tar for seg å utvikle en pakke til spillmotoren Unity som lar programmet kommunisere med en SensMax TAC-B 3D-W mennesketellende radarsensor. Vi søkte om å få oppgaven i oppgavefordelingen organisert av NTNU da vi ville jobbe med noe innen hardware/IoT, og fikk deretter tildelt denne fra NTNU.

1.2 Prosjektbeskrivelse og problemstilling

Able Magic AS er et selskap som er basert i Trondheim som "lager ulike former for historiebaserte opplevelser, fra utstillingsdesign og interaktive installasjoner til spill, interaktive bøker og andre applikasjoner for mobile plattformer". I sine prosjekter bruker de ofte sensorer av diverse slag for interaktivitet, og prosjektene er ofte bygget i Unity. Dette danner grunnlaget for oppgaven, som er å utvikle en utvidelsespakke til Unity som har som hensikt å la Able Magic AS eller andre aktører ta i bruk SensMax-sensoren i senere prosjekter.

Problemstillingen gruppen skal utforske i sammenheng med ovennevnte prosjektbeskrivelse er å utforske diverse måter å bruke sensoren de får utdelt på, det er formulert som følger:

"Hvilken bruksområder kan en mennesketellende radarsensor brukes på i spillmotorer for å lage interaktive utstillinger og spill?"

1.3 Resultatmål

Gruppen skal produsere et solid API som har få feil og uoverensstemmelser, og som fungerer som forventet. Dette API-et skal ha god og tilgjengelig dokumentasjon som gjør det enkelt for sluttbruker å sette seg inn i funksjonaliteten, eller for utviklere å opprette ny funksjonalitet eller fikse feil. Produktet skal ha åpen kildekode på nettet hvor andre kan bygge videre på pakken ved senere anledning for å utvide funksjonaliteten.

Det skal produseres minst et eksempel på hvordan API-et kan brukes i Unity. Dette kan være et enkelt spill eller annen programvare som er relevant for oppdragsgivers behov. Dette vil også fungere som en demonstrasjon for oppdragsgiver.

Oppdragsgiver har kjøpt inn to sensorer, som de kan bruke videre etter prosjekt perioden. Det er et mål at det ikke skal påløpe flere kostnader med dette prosjektet. Dette skal

være mulig da prosjektet er programvare utvikling å ikke skal påløpe flere kostnader og arbeidskostnader ikke tenkt.

Under prosjektperioden skal gruppen samarbeide om å opprettholde alle frister som blir pålagt dem. For å oppnå dette skal skriftlige innleveringer startes i god tid, og at både dokumentasjon og kode jobbes med parallelt.

1.4 Effektmål

I løpet av prosjektet skal en Unity-pakke utvikles. Det er et mål for gruppen at denne skal være nyttig for oppdragsgiver og eventuelt kunne brukes av andre som ønsker å benytte SensMax Tac-B sensoren i sine Unity-prosjekter.

Gruppen vil utforske måter å bruke sensoren i utvikling av spill og andre interaktive opplevelser og håper å finne innovative måter å benytte utviklet programvare.

Prosjektetet skal være kostnadseffektivt for oppdragsgiver og gruppe medlemmer. Da programvaren blir tilgjengelig som åpen kildekode vil prosjektet ikke bli lønnsomt, men det er et mål at det skal koste så lite som mulig.

At resulterende rapporter, kode og presentasjoner skal være av en kvalitet som møter vurderingskriteriene for en god oppgave.

1.5 Rammer

Det er ingen spesielle krav til oppgaven med tanke på penger og tid. Sensor kreves under utvikling, i forhold til dette kreves også et rom der sensor kan monteres for testing. Oppdragsgiver låner studentene sensorer under arbeidet, og beholder disse etter endt oppgave. Studentene har tilbud om kontorplass hos oppdragsgiver.

3 Gjennomføring

3.1 Hovedaktiviteter

Studentene vil gjennomføre følgende hovedaktiviteter. Alle aktiviteter kan foregå både på egenhånd og sammen, med unntak av planlegging som stort sett er en felles oppgave;

- Lesing, læring og oppslag: underveis i prosjektet vil det kreves mye egenlæring; både med hensyn på oppgaven og prosjektet. Dette inkluderer å lese dokumentasjon av nødvendig programvare og APIer slik at og lesing av relevante artikler og bøker. Informasjon kan letes opp på søkemotorer via internett, ved hjelp av biblioteket eller andre steder der det kan være relevant, men det er viktig at å bruke troverdige kilder.
- Skriving: under hele prosjektet skal det jobbes jevnt på hovedoppgaven for å holde oversikt over tid og arbeid som gjenstår. I tillegg skal det gjennomføre noen obligatoriske innleveringer, henholdsvis denne forprosjektsplanen og en posterpresentasjon i uke 13. Videre skal det også skrives en god del dokumentasjon om prosjektet, slik at det er mulig å utvikle videre eller ta i bruk prosjektet ved senere anledning. Dette er hensiktsmessig å gjøre underveis mens man ennå husker hvordan og hvorfor ting er utviklet som de er, men det er ingenting i veien for å skrive etter ferdigstilling av selve koden.
- Planlegging: i begynnelsen av prosjektet og i starten av sprintene skal det planlegges godt, for å skape og holde oversikt over hva som må gjøres og til hvilken tid. Dette skal føres i referater, diagrammer, kanban, lister eller andre relevante dokumenter slik at det kan sees tilbake på ved senere anledning, for eksempel under sprint reviews.
- Møter: gjennom hele oppgaveperioden skal det organiseres jevnlige møter med veileder og oppdragsgiver. Henholdsvis skal møter holdes cirka hver tredje uke med veileder, for å få oppklaring og veiledning og å rapportere fremdrift, og oppdragsgiver holdes hver andre uke for å forsikre at utviklingen går i ønsket retning. Disse møtene vil fungere som "sprint reviews". Veileder vil være til stede på noen av disse. Møter dokumenteres via møteinnkallinger med agenda, og møtereferater skrevet av studentene dokumenterer møtene og beslutninger i etterkant.
- Dokumentasjon av arbeid: det er viktig å føre timeliste underveis, slik at vi ser hvor mye tid vi har igjen, hvor tiden går, og som en dokumentasjon på hva vi har gjort når. Dette gjøres i regneark lastet opp i Microsoft Teams.
- Koding: fra og med uke 5 begynner studentene å kode. Gruppen jobber skal i all hovedsak jobbe i 2-ukers sprinter inspirert av agil utviklingsmetodikk, med sprintplanlegging og sprintgjennomgang henholdsvis før og etter hver enkelt sprint. De

forskjellige sprintene har ofte forskjellige overordnede mål allerede nå. Noen eksempler er; MVP-utvikling, demonstrasjonsprosjekt-utvikling og brukertest-utvikling. Mål kan selvsagt endres underveis, dersom ikke alt går som planlagt. Koding skal etter planen være ferdig i uke 13.

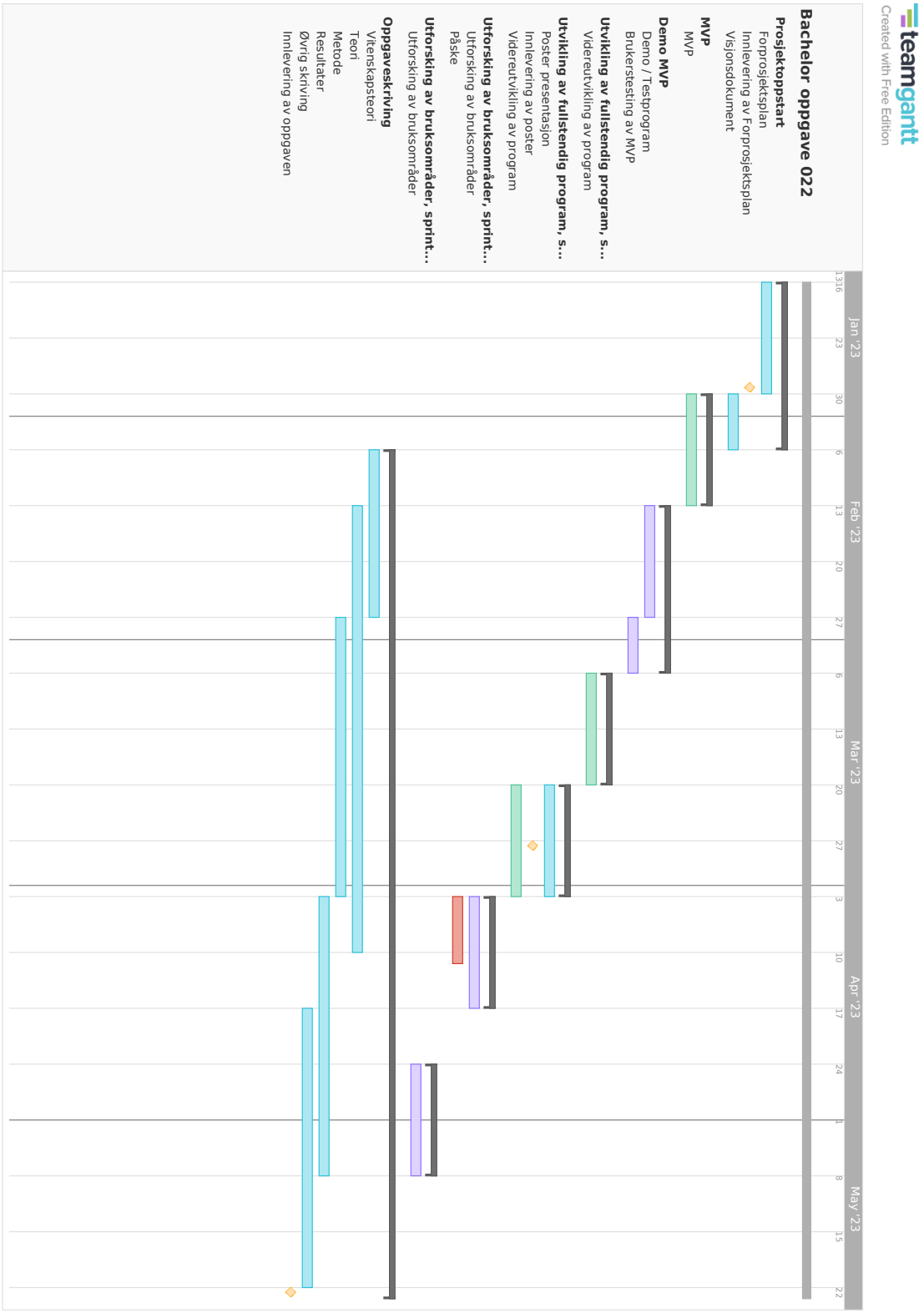
- Brukertesting: i samsvar med Gantt-diagrammet skal det foregå brukertester i ukene 14-18, utenom i påsken. Her ønsker studentene å samle mye informasjon om hva slags bruksområder som er gode, interessante eller engasjerende for sensoren, og på hvilken måte (på gøy, for trening, etc.). Denne informasjonen trengs til å skrive ferdig oppgaven, og til å fikse eventuelle feil og mangler i koden. For å utføre brukertestene kreves minst en MVP av prosjektet, og et eksempel på hvordan det kan brukes. Helst skal prosjektet være tilnærmet ferdig, og flere eksempler er klare til testing.

3.2 Milepæler

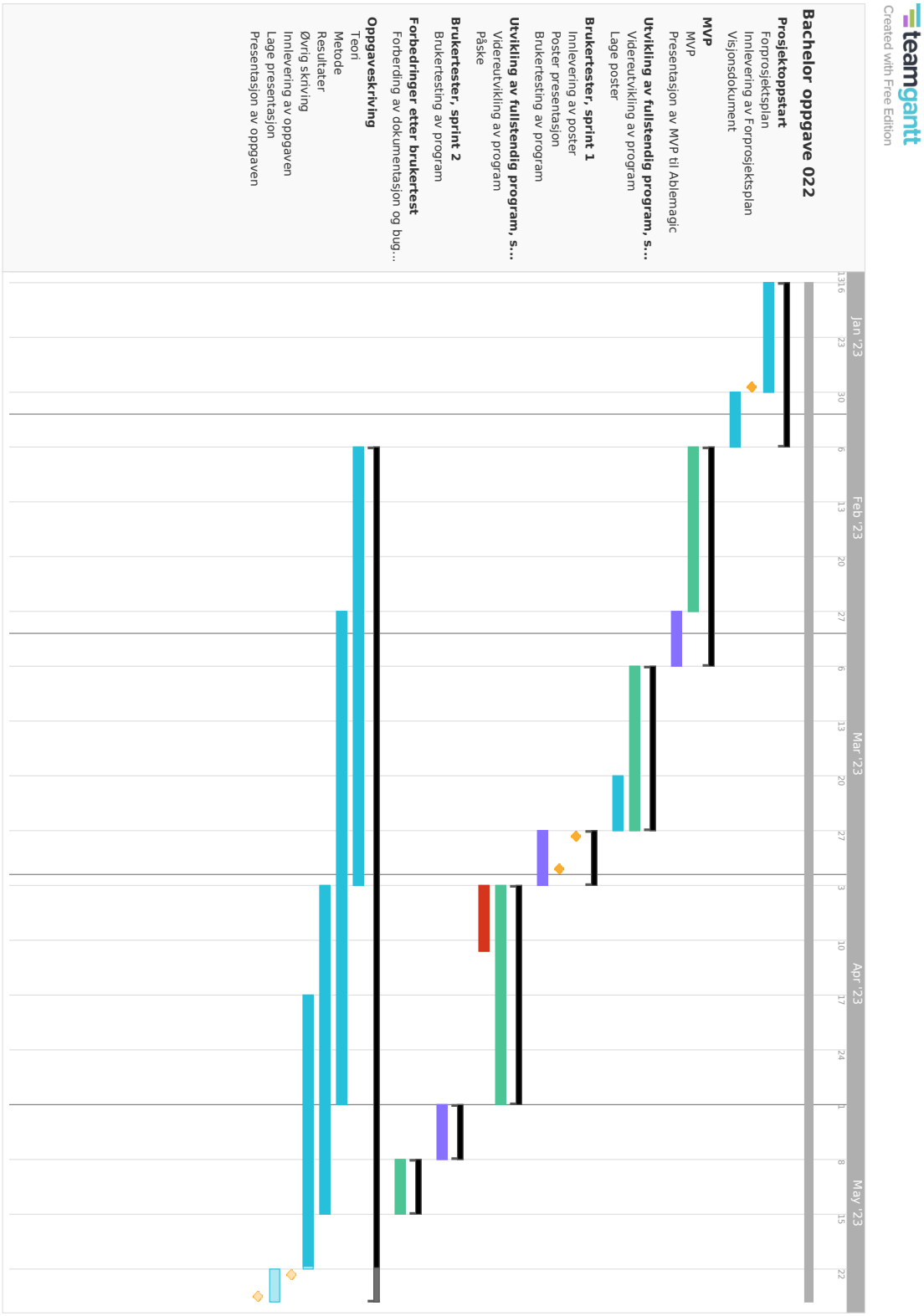
Dato	Milepæl
27.01	Innleveringsfrist for forprosjektsplan
03.02	Visonsdokument klart
24.02	MVP og demonstrasjon klar til testing
27.03	Innlevering av poster
31.03	Ferdigstilling av prosjektkoden
05.05	Testing av bruksområder ferdig
22.05	Innleveringsfrist på hovedrapport og prosessdokumentasjon
26.05	Presentasjon av oppgaven

Tabell 1: Milepæler

B.1 Gantt diagram



B.2 Faktisk gjennomføring av prosjektet



C Visjonsdokument

4.4 Sammendrag av brukernes behov

Behov	Prioritet	Påvirker	Foreslått løsning
Motta posisjonsdata fra sensoren	Høy	Data-innhenting	Mellomvare som håndterer dette
Pakken er lett å bruke i prosjekter	Høy	Bruker-vennlighet	En Unity pakke som skal enkelt kunne kobles til sensorens webgrensesnitt
Godt dokumenterte metoder som er enkle å forstå hva gjør	Høy	Bruker-vennlighet	En Unity pakke som er godt dokumentert så bruker enkelt forstår hvilke metoder som gjør hva
Posisjonsdata kan omregnes fra relativ posisjon i forhold til sensoren til kartesiske koordinater i rommet	Høy	Data-håndtering	Mellomvare som håndterer dette
Koblingen til serveren er stabil under bruk	Høy	Data-innhenting	MQTT-server vil kjøre som del av brukerens Unity-prosjekt og dermed være like stabil som prosjektet
Motta bevegelsesdata fra sensoren	Medium	Data-innhenting	Mellomvare som håndterer dette
Lett å navigere dokumentasjon på nett	Medium	Bruker-vennlighet	GitHub Wikien vil tilgjengeliggjøre dokumentasjonen
Oppsett av posisjonen til sensoren og andre data for omregninger er enkelt, men ikke nødvendig dersom bare rådata er ønsket	Lav	Bruker-vennlighet	Dette er en valgfri del av oppsettet til pakken

Tabell 5: Brukernes behov

4.5 Alternativer til vårt produkt

Per 03.02.2023 finnes det ingen gode alternativer til vårt produkt på markedet. Det er mulig å lage sin egen løsning i Unity for en utvikler, og utviklere vil ha kunnskapen til å gjøre dette. Løsningen utviklere lager selv vil være mer rettet mot deres behov, men vil ta tid og energi å lage.

Prosjekt 22

6 Produktets funksjonelle egenskaper

Pakken har relativt enkle funksjonelle krav da fokuset i prosjektet i stor grad er på de ikke funksjonelle kravene som dokumentasjon og brukervennlighet. Under er en liste av de funksjonelle kravene som forutsatt av gruppen ved prosjektstart.

- MQTT-server som kommuniserer mellom Unity og sensoren
- Pakken kan bli installert i alle typer Unity-prosjekt
- Funksjonalitet for å konvertere JSON-data fra sensoren til datastrukturen vi velger å bruke
- Funksjonalitet for å konvertere koordinat-data fra sensoren til koordinat-data som kan brukes i Unity, for eksempel til å posisjonere spill-objekter som spillerens avatar
- Funksjonalitet for å lese bevegelsesdata fra sensoren
- Server starter når Unity-prosjektet starter
- Mulighet for å kommunisere med flere sensorer samtidig
- Funksjonalitet for å registrere sensorens plassering i spillmiljøet med id
- Funksjonalitet for å bestemme hva pakken skal lese av informasjon fra en gitt sensor (posisjonsdata, bevegelsesdata, begge)
- Funksjonalitet for å bare få informasjon om det er en person i en gitt sone fra sensor
- Funksjonalitet for overvåkning av hvor mange mennesker som er i et bygg/rom ved hjelp av en eller flere sensorer
- Støtte for begge størrelsene pakke sensoren kan sende og automatisk deteksjon dette

Prosjekt 22

7 Ikke-funksjonelle egenskaper og andre krav

- All funksjonalitet som brukere kan ta i bruk må være godt dokumentert. Dette er for at brukere enkelt kan forstå hvilken funksjonalitet pakken har, og for at de skal ta denne i bruk på en god og effektiv måte med lite friksjon.
- API-et vi utvikler må være godt testet slik at det oppfører seg som forventet fra brukerens side. Vi vil sikte på å høy testdekningsgrad der vi ser at dette er hensiktsmessig.
- Produktet skal være responsivt, og gi god tilbakemelding til bruker når ting laster eller av andre grunner stopper opp.
- Produktet skal gi god beskjed dersom feil eller mangler oppstår, og loggføre dette slik at det kan sendes til utvikler slik at dette kan rettes opp i.
- Produktet skal ha et oversiktlig brukergrensesnitt som er klart, tydelig og lett å forstå.
- Brukergrensesnittet skal følge WCAG 2 der det er mulig og/eller hensiktsmessig.
- Produktet skal være lett å videreutvikle, slik at andre utviklere kan legge til funksjoner eller fikse feil dersom de ønsker det.
- Produktet skal være kompatibelt med så mange Unity-versjoner som mulig, men "Long Term Support"-versjoner (LTS) skal være prioritert.
- Produktet skal virke som forventet på Windows, MacOS og Linux.
- Produktet skal være trygt, og ikke skape flere sikkerhetshull enn de sensoren eventuelt har fra før.
- Produktet skal være modulært og tilby bruker muligheten til å legge til ikke essensiell funksjonalitet kun ved behov.

D Detaljerte resultater fra brukertester

D.1 Brukertest 1

Brukerenes selvbeskrevede erfaringsnivå. Ikke direkte sitat.

Bruker	**Generell kodeerfaring**	**C#-erfaring**	**Unity-erfaring**	**MQTT-erfaring**
Fredrik	God erfaring	Over 100 timer erfaring	Nominert til Game of the Year i Norge	Ingen erfaring
Anonym	Litt erfaring; har hatt faget "Informasjonsteknologi, grunnkurs" og har programmert en del, men studerer ikke IT	Litt erfaring med enkle ting	Jobbet litt med et enkelt prosjekt; 1/10	Har hørt navnet, men vet ikke hva det er
Felix	Veldig erfaren	Ikke så erfaren, fortrekker å unngå Microsoft-produkter	Vet hva det er; har sett og installert det, men aldri brukt	Vet at det brukes til IoT
Tore	Veldig erfaren; går 5. året på datateknologi hos NTNU og er god til å kode	Et halvt år med erfaring	Et halvt år med erfaring; har vært med å vinne Norwegian Game Awards	Har nylig hørt om det, vet det brukes til IoT
Petter	Har programmert en del, er ikke først og fremst programmerer	Har jobbet mye med C# i arbeidet, men kun hatt opplæring i Python og Matlab gjennom skole	Begynte å bruke Unity litt av og på etter han startet i jobb hos Ablemagic for 2,5 år siden	Har hørt om MQTT, men ikke brukt det, vet at det er en IoT protokoll

Tabell 1: Brukertestdeltagerne sitt erfaringsnivå i relevante teknologier

Oppgave 1: Du skal lage et Unity-prosjekt som benytter seg av Sensmax TAC-B sensoren og du vet at det finnes en pakke for dette. Implementer den i dette prosjektet.

Bruker	**Oppgave løsning**
Fredrik	Lette først etter en GitHub release. Lastet deretter ned som pakke og pakket ut. Forsøkte å importere som "new Asset" i Unity. Trengte hjelp til å bruke Unity sin pakkebehandler. Sier Asset er mer vanlig.
Anonym	Brukte pakkebehandleren og importerte fra Git på riktig måte, men fikk en feilmelding på grunn av at GitHub hadde endret RSA SSH host key dagen før. Testadministrator hadde ikke kunnskap om problemet SSH-problemene enda, men hjalp til med å legge til pakken lokalt slik at testen kunne fortsette.
Felix	Leste først litt i README, kommenterte at han likte at prosjektet benytter MIT-lisens. Gikk deretter inn i innstillingene til pakkebehandleren, før han fant selve pakkebehandleren med et dytt i riktig retning. Han la til pakken via Git, men fikk feilmelding pga SSH-problemene nevnt tidligere. Han fikk lagt til pakken med HTTPS i steden uten problemer.
Tore	Sjekket med en gang README. Fant deretter pakkebehandleren. Forsøkte med URL til repoet først, men når det ikke fungerte brukte han HTTPS. Ville helst bruke SSH, men kunne ikke det på grunn av problemer nevnt over.
Petter	Ville laste ned pakken. Har ikke mye erfaring med pakker. Fikk tips om README, og begynte der. Åpner pakkebehandleren etter å ha lest. Trykte på importer fra Git. Prøver URL fra nettleseren, men får et hint om å bruke URL som slutter på ".git". Importerer uten problemer.

Tabell 2: Resultater for oppgave 1 i første brukertest

Oppgave 2: For å kunne kommunisere med sensoren må det settes opp administrasjon av MQTT-protokollen, fullfør oppsett av en manager og kjør programmet.

Bruker	**Oppgave løsning**
Fredrik	Fant ut av "prefabs" ganske greit, drar manager inn. Kjører uten problemer.
Anonym	Spurte om det står i README. Leste deretter i README-filen. Finner pakken i Unity uten problem. Drar inn prefab uten problem. Kjører uten problem.
Felix	Leste i README. Fikk problemer med konsollen. Fant fram i mappestrukturen og fikk lagt inn manager og sensor uten problemer. Fikk litt hjelp til å sette seg inn i Unity sitt brukergrensesnitt.
Tore	Leste ved et uhell oppgaven på egenhånd mens referat ble skrevet, gjennomførte før testadministrator rakk å fortelle oppgaven. Fikk inn manager uten problem.
Petter	Leste README slik oppgaven etterspør. Fant pakken i Unity enkelt. Dro inn manager uten problem. Leste mer i README. Dro sensor på manager slik at den blir barn i hierarkiet, som egentlig er en senere oppgave.

Tabell 3: Resultater for oppgave 2 i første brukertest

Oppgave 3: Siden du får en feilmelding, ønsker du å feilsøke klienten. Endre manageren slik at klienten logger all aktivitet.

Bruker	**Oppgave løsning**
Fredrik	Stoppet spillmodus, fant innstillingen og endret den uten problemer.
Anonym	Prøvde å endre mens Unity var i spillemodus. Ble bedt om å stoppe spillemodus. Husket å justere innstillingene før han startet spillmodus på nytt.
Felix	Trykket på manager og satte loggføringsnivå uten problemer.
Tore	Gjennomførte uten problemer. Synes oppgaven var litt for lett til å være på testen. Endret på både klienten og server, selv om oppgaven bare etterspurte endring av klienten.
Petter	Stoppet spillmodus. Endret loggføringsnivå uten vanskeligheter. Starter spillmodus.

Tabell 4: Resultater for oppgave 3 i første brukertest

Oppgave 4: Nå som manageren er klar ønsker du å håndtere innkommende meldinger fra sensoren med serienummer "051001572". Legg til sensoren sånn at den kan få pakker. fra 051001572

Bruker	**Oppgave løsning**
Fredrik	Legger sensor feil, ikke som barn av manager, men utenfor. Putter serial number riktig. Sjekker feilmeldinger, antar alt er good. Etterspør at default manager burde ha en sensor. Foreslår at Unity.GUIDrawer kan brukes til å legge til knapp som kan legge til en sensor på manager inspektor vinduet.
Anonym	Sjekker README. Legger til sensor som barn av manager. Skriver inn serienummer uten problem.
Felix	La inn serienummer uten problem, hadde alt lagt inn sensor i oppgave 2.
Tore	Sjekket ikke readme og la ikke inn sensoren som barn av manageren først. Etter å ha sett at feilmeldingen fortsatt kom sjekket han readme og fikset problemet uten behov for hjelp.
Petter	Dobbeltklikker instinktivt på feilmelding som åpner Visual Studio. Skriver inn serienummer på riktig sted. Sensor var alt lagt inn tidligere. Kjører Unity.

Tabell 5: Resultater for oppgave 4 i første brukertest

Oppgave 5: Sensor spillobjektet lagrer data fra sensoren, for å benytte denne i spillverden legg til en sample-skript som lager spillobjekter som representerer folkene sensoren detekter på sensor objektet.

Bruker	**Oppgave løsning**
Fredrik	Leser README igjen, drar skript på sensoren som README sa. Dobbeltsjekker README. Populerer GameObjects i alle tomme slots av animasjon sample kode uten problem.
Anonym	Finner fram til skriptene uten problem, leser README. Drar skript på sensoren uten problem. Får litt hjelp med å lage et 3d objekt å legge til på skriptet. Får alt til å virke.
Felix	Sjekker README. Finner fram uten problem. Legger til skript uten problem. Får litt hjelp til å lage en et 3D objekt. Dro den oppå sensor som et barn først, fikk hjelp med å legge det til på skriptet.
Tore	Begynte på å lage et nytt skript selv, siden dette var utenfor testen rammer, stoppet testadministator han og hintet om å sjekke readme igjen. Finner da samples og legger til det.
Petter	Stopper Unity. Leter litt rundt i mappene selv før vi minner på README. Dro skripter på uten problem, fikk litt hjelp å lage en prefab å dra på. Skript funker uten problem.

Tabell 6: Resultater for oppgave 5 i første brukertest

Spørsmål 1: Hvordan synes du det var å sette opp pakken?

Bruker	**Svar**
Fredrik	Han beskriver at han foretrekker å importere pakker som assets med fra release. Dette forutsetter at det er en release på repoet som er i Unity-Package filformatet.
Anonym	Litt problemer med Git. Ville nok klart å finne ut av noe selv, men det er ganske vanskelig spørsmål. Hadde ikke forstått med en gang men hadde nok forstått etter hvert.
Felix	Helt fint, å legge til pakken med Git-url virker som en bra standard.
Tore	Veldig greit, godt forklart av dokumentasjonen. Ville ha satt seg mer inn i skriptene hvis han faktisk skulle jobbe med pakken.
Petter	Veldig greit, enkelt å bruke package manager. Likte å sette opp prefabs, og at det kan legges til flere sensorer. Virker nyttig. Og at fikk opp i editor hvilket serienummer som manglet som gjorde det lett å registrere sensor. Føltes plug and play.

Tabell 7: Brukeres tanker om enkelheten til oppsett av pakken

Spørsmål 2: Hvordan synes du det var å finne frem til skriptene du trengte?

Bruker	**Svar**
Fredrik	Synes sensor/manager relasjonen var litt forvirrende. Ønsker at sensorer blir generert automatisk.
Anonym	Synes det gikk veldig fint å finne frem.
Felix	Fant fort til å ikke være vant til Unity. Synes pakkestrukturen er fin.
Tore	Det var greit med readme, men synes det er rart at samples lå i runtime.
Petter	Gikk greit, måtte bare bli kjent med strukturen. Men gikk veldig fort.

Tabell 8: Brukeres tanker om filstrukturen til pakken

Spørsmål 3: Hvordan var helhetsopplevelsen av pakken?

Bruker	**Svar**
Fredrik	Syns det var greit, likte godt eksempel koden. Synes alltid det er litt rotete å importere assets. Etterspør flere muligheter for importering.
Anonym	Sier pakken er grei, men endrer til bra. Synes han ble litt påvirka i positiv retting av at han fikk hjelp. Han kunne ha likt den mindre hvis han ble stående fast på en oppgave lengre.
Felix	Ganske bra. Helt fint. Eneste problemene var med Unity selv. Enkelt å bruke.
Tore	Virka grei, har ikke testet påliteligheten. Virka relativt enkelt å bruke til å styre spill objekter.
Petter	Veldig bra. Virka veldig greit. Hvilken funksjonalitet som finnes og sånn hadde han antagelig funnet selv dersom vi ikke var der. Bra helhetsinntrykk.

Tabell 9: Brukeres helhetsopplevelse i første brukertest

D.2 Brukertest 2

Brukerens selvbeskrevede erfaringsnivå på en skala fra 1-5 der 1 er lite og 5 er mye erfaring.

Bruker	**Generell kodeerfaring**	**C#-erfaring**	**Unity-erfaring**	**MQTT-erfaring**
Aleksander	3	3	2, brukt for mange år siden	Ingen erfaring
Peder	5	5	4	Ingen erfaring
Anonym	4-5	4-5	4-5	Ingen erfaring
Matilde	4	3	3	Ingen erfaring
Trym	3-4	2-3	2-3, var veldig god på det for noen år siden	Ingen erfaring
Rinsai	3	3	5	Ingen erfaring

Tabell 10: Testbrukeres selvbeskrevet erfaingsnivå i relevante teknologier

Brukerens selvbeskrevede erfaringsnivå på en skala fra 1-5 der 1 er lite og 5 er mye erfaring.

Bruker	**Generell kodeerfaring**	**C#-erfaring**	**Unity-erfaring**	**MQTT-erfaring**
Aleksander	3	3	2, brukt for mange år siden	Ingen erfaring
Peder	5	5	4	Ingen erfaring
Anonym	4-5	4-5	4-5	Ingen erfaring
Matilde	4	3	3	Ingen erfaring
Trym	3-4	2-3	2-3, var veldig god på det for noen år siden	Ingen erfaring
Rinsai	3	3	5	Ingen erfaring

Tabell 10: Testbrukeres selvbeskrevet erfaingsnivå i relevante teknologier

Oppgave 1: Du har installert en pakke i Unity for å kommunisere med SensMax TAC-B sensoren. Du har lagt inn en manager som administrerer sensorer. Legg til en ny sensor til denne manageren.

Bruker	**Oppgave løsning**
Aleksander	La først til sensor-skriptet på manageren. Lette i filmappen, etter en prefab. Fikk litt hjelp av testadministrator til å se på inspector-vinduet. La til sensor scriptet på manager et par ganger. Ble oppfordret til å sjekke README. Fant deretter knappen og la til sensoren.
Peder	Gikk inn i README, lurte på om det er en tekst som forklarer hva han må gjøre. La først til sensor-skriptet på managern. Prøvde å legge inn ny manager, men innså at den alt var der. Slo på logging. Trykket tilslutt på "add sensor"-knappen.
Anonym	Sjekket README. Brukte tid på å finne riktig del. Laget sitt eget GameObject med script som var en alternativ løsning.
Matilde	Trykket intuitivt på sensor objektet. Så på manager. Gikk deretter inn i README. Leste tilsynelatende hele README (Sa etter testen at hun leste hele README 2 ganger). Trykker på sensor igjen. Fant add sensor knappen på managren.
Trym	Startet med å lese igjennom README. Kommenterer at det virker “straight forward” med prefabs som i “vanlig unity”. Fortsetter å lese. Innser at vi har satt opp en manager alt. Fant så “Add Sensor”-knappen enkelt.
Rinsai	Spurte om vi hadde API-doc, som ble vist. Fant veldig lett knappen.

Tabell 11: Resultater for oppgave 1 i andre brukertest

Oppgave 2: Du har lyst til å ha brukernavn og passord på manageren og sensoren din. Sett opp manageren til å bruke brukernavnet "test" og passordet "test".

Bruker	**Oppgave løsning**
Aleksander	Fikk hjelp til å legge filer i assets mappa av testadministrator. La inn brukernavn og passord som READMEen forteller. Leste videre i README om hvordan han legger til path. Gikk inn i manageren, fant secret file path. Skrev inn fil-path uten problemer.
Peder	Sjekket først manager-objektet. Åpnet så README, fant enkelt riktig seksjon. Kopierer JSON utsnitt. Åpner filutforsker i Windows. Lager en tekstfil i assets. Limer inn JSON. Leser på README igjen, om path to file. Kopierer path fra filutforsker og limer inn på riktig sted. Den fant ikke fila. Redigerer til relativ path. Unity fant fortsatt ikke filen pga skrivefeil i path.
Anonym	Klikket på manager, la merke til "Add Sensor"-knappen og kommenterte på forrige oppgave. Lette også på sensoren før han gikk inn i README. Fant enkelt riktig seksjon. Kommenterte: "Sier ikke noe om filtype". Går inn i filutforsker. Lager tekstfil. Spør om hvordan relativ path er i Unity og får avklart. Navnga filen med .txt i tillegg til automatisk filtype, måtte fikse filnavnet for å få riktig path.
Matilde	Trykket på sensoren for å lete etter passord/brukernavn-felt. Gikk inn i README og leste riktig seksjon. Spurte om hjelp med filplassering og fikk forslag av testadministrator. Laget skript-fil i Assets. La inn JSON-strengen i C# fil. Ble hjulpet med å åpne filutforsker på riktig sted. Laget først Word dokument etter å ha spurt om filtype har noe å si og fått beskjed om at filleser bør tåle det meste. Ombestemte seg og laget txt-fil. Limer inn JSON fra README. Så etter variabel i sensor først, men fant riktig den i manager uten problem. Limer inn path. Fikk compiler error pga skriptfilen hun lagde. Slettet overflødige filer.
Trym	Hadde sett seksjonen i README i tidligere oppgave. Spurte om hvor han skal plassere filen, og hvordan man lager fil. Laget C# skript og limer inn JSON der. Spurte om .cs funker, får vite at det vil gi til kompileringsfeil. Slettet en ekstra kopi av C#-fila. Laget en tekstfil istedet. Kommenterer at det er "litt IQ-test". Leser om Secret File Path i README. Fant riktig sted å putte inn, men skrev bare filnavn. Fikk lagt inn path riktig, etter feil ble oppdaget.
Rinsai	Gikk inn i README og fant informasjon om credentials. Var usikker på hvilket filformat, antok txt-fil. Lager en tekstfil i Assets, limer inn JSON riktig. Var usikker på relativ/full path. Leste i koden for å finne ut.

Tabell 12: Resultater for oppgave 2 i andre brukertest

Oppgave 3: Du ønsker å loggføre statistikk fra sensoren, og vet at den sender pakker med informasjon hvert 5. minutt. Legg til funksjonalitet for å loggføre disse pakkene.

Bruker	**Oppgave løsning**
Aleksander	Fant riktig del av README og leste den, fant riktig fil. Sjekket README for hvordan han legger til funksjonalitet. Brukte "add component"-knappen han hadde brukt i første oppgave til å legge til skriptet.
Peder	Leste i README om loggføring av 5 min pakker. La til riktig skript fra Inspector GUI ved å søke det opp. Laget ny tekstfil for logging og skrev inn path.
Anonym	Leste om 5 min pakker i README. La til skriptet på et nytt GameObject. Spurt om han måtte lage en fil. Dette gir feilmelding, da skriptet bare fungere dersom det er på manager-objektet. Bruker oppdaget ikke feilen og sa seg ferdig med oppgaven.
Matilde	Leste i README. Var usikker på hvor skriptet skulle legges til, men la det til på manager. Gjorde det helt korrekt, men er usikker på om hun må skrive inn en file path. Sjekket README. Antok hun må lage en fil, og gjorde dette uten problem.
Trym	Hadde sett seksjonen i README allerede. Tror vi mente debug-logging på manager. Fant riktig skript når han leste litt mer. Dro den på manager. Var litt forvirret om forskjellen på logging til konsoll og logging til fil.
Rinsai	Husket fra README. Søkte opp DataPacketClient og la til på manager uten problem. Laget ny folder i Assets kalt Log. La inn path til folderen. Etter å ha lest kode laget hun en fil og la inn path til den. Kjørte fint. Ikke åpenbart for henne at default lager fil.

Tabell 13: Resultater for oppgave 3 i andre brukertest

Oppgave 4: Du skal lage et skript som oppdaterer posisjonen til et objekt hver gang ny data mottas fra sensoren. Hvordan ville brukt pakken vår til å oppnå dette? (Det er ikke nødvendig å faktisk lage et script)

Bruker	**Oppgave løsning**
Aleksander	Gikk inn i assets og laget et skript. Er stuck, vet ikke hvor han ville begynt. Får tips om å sjekke ut samples. Prøvet seg først med Status Change Event, men vi er ute etter New Message Event.
Peder	Ble forvirret av oppgave fremleggelsen og så først etter en metode. Leser usage-seksjonern i README igjen. Han er litt usikker på hvor han skal gå herfra. Åpner Sensor.cs i Visual Studio, leser igjennom. Åpner Sensor.cs i VS Code, prøver å åpne Entity.cs men diverse urelevante ting dukker opp. Bruker til slutt VS igjen. Fikk oppklaring av av oppgaven, og skjønner han at skal bruke UnityEvents. Han sjekker Sensor.cs for hvordan, leser koden for å finne ut hvordan.
Anonym	Så på SpawnOnEntity-skriptet for å finne entrypoint. Sa han ville etterlignet dette skripet. Så litt videre og er usiker på hva oppgaven er ute etter. Repeterte ville etterligna SpawnOnEntity-skriptet. La så merke til at vi bruker Listener og finner eventen, men kommenterer at han ønsker å vite hva eventen sender.
Matilde	Sjekket README. Åpnet wiki, men velger å gå tilbake til README med en gang. Åpnet samples på github. Går tilbake til README. Er usikker på målet med oppgaven og får oppklaring. Hun åpnet og leser hele SpawnOnEntity-skriptet. Leser så hele Sensor-skriptet. Er usikker på hva oppgaven er ute etter. Hun leser nøye koden til HandleMessage-metoden. Trykker messagereceivedevent, og stusser litt på hvordan Eventet funker, men sier hun ville nok brukt den.
Trym	Trengte litt mer forklaring hva vi var ute etter. Han vil starte med å se hvordan vi får inn meldinger. Ser igjennom datapcaketclient sin kode. Misforsto, trodde vi mente hver gang vi fikk 5-min meldinger, så vi forklarete at vi mente sensormeldinger. Han begynner å se igjennom sensor.cs. Han ville ha lett igjennom koden og sett om det var en metode som gir payload. Kommenterte at "kan hende at fra [messageRecieved-eventen] så får han noe info".
Rinsai	Så etter en Event i editor UI. Sier at vi ikke har en event å lytte til. Søker i koden til manager etter received, leter videre i koden etter noe å koble på. Får tips om å fokusere mer på sensor skriptet. Sa hun misforsto oppgaven, og trodde det var generelt for alle meldinger og dermed i manager. Fant event i sensor når hun skjønte det var en spesifikk sensor.

Tabell 14: Resultater for oppgave 4 i andre brukertest

Oppgave 5: Du skal lage et skript som bare benytter seg av objektet nærmest sensoren. Hvilken metode/funksjon ville du brukt?

Bruker	**Oppgave løsning**
Aleksander	Får litt hjelp av Alida til å sjekke ut dokumentasjon. Uheldigvis står GetClosestEntity åpen og han ser den med en gang.
Peder	Er innstilt på UnityEvents og leter etter riktig event. Tenker igjennom hva som er i Entity. Stopper ved DistanceFromParent, og tenker at det bare er å gå igjennom alle entities og bruke den som er kortest unna.
Anonym	Går inn på Event-greiene for å se om de har info om distanse. Leser i Sensor.cs om Eventene. Ville brukt MessageReceivedEvent. Leter etter en definisjon/innhold av hva som er i Eventet. Lurer på om innholdet av Eventet er i dokumentasjon. Ville regna det ut selv. Leter etter hvert i dokumentasjonen.
Matilde	Leter videre i Sensor.cs. Fra toppen. Leser docs til HandleMessage igjen. Scroller sakte men sikkert nedover. Finner GetClosestEntity.
Trym	Har allerede sett getclosestentity i tidligere oppgave og svarer at han ville brukt den.
Rinsai	Skjønte ikke oppgaven helt. Tenkte først at å justere sensor sit range of view vil få den til å bare detektere ting innen for range of view. Lette mye gjennom Editor GUI (inspector). Åpnet koden igjen etter å ha sett gjennom alt i Unity GUI. Så etter variabel som kan hjelpe, og fant Entities. Så etter hva slags info "Entity" inneholder. Fant til slutt rett metoden.

Tabell 15: Resultater for oppgave 5 i andre brukertest

Spørsmål 1: Hvordan synes du det var å sette opp og konfigurere sensorer?

Bruker	**Svar**
Aleksander	Hvis jeg hadde hatt tid og ingen som så på og dokumentasjonen skulle jeg fått det til. Og man må kunne litt Unity.
Peder	Litt handicap siden han ikke visste hva pakken gjorde, skulle brukt mer tid på å lese. For det står hva slags sensor det er, men han vet ikke det. Han antar en person som vet hva han skal vet bedre, at han gikk inn litt for blind. Når han satt seg inn i det gikk det greit. File paths var litt forvirrende mtp om det skulle være i relativ eller fullpath.
Anonym	Både manuelt og med knapp virka relativt greit. Er ærlig og sier at å skrive en egen fil for credentials var litt herk. Ville heller hatt felter i Unity dersom det er trygt/mulig.
Matilde	Syns det var veldig greit å legge til sensoren. Veldig lett.
Trym	Veldig greit. En knapp bare. Enkelt å finne. Har glemt litt Unity, men det var straightforward.
Rinsai	Enkelt. Ingen problemer. Lurte på filformatet til logging bare.

Tabell 16: Brukers opplevelse av sensoroppsett

Spørsmål 2: Hvordan synes du det var å finne frem til scriptene og dokumentasjonen du trengte?

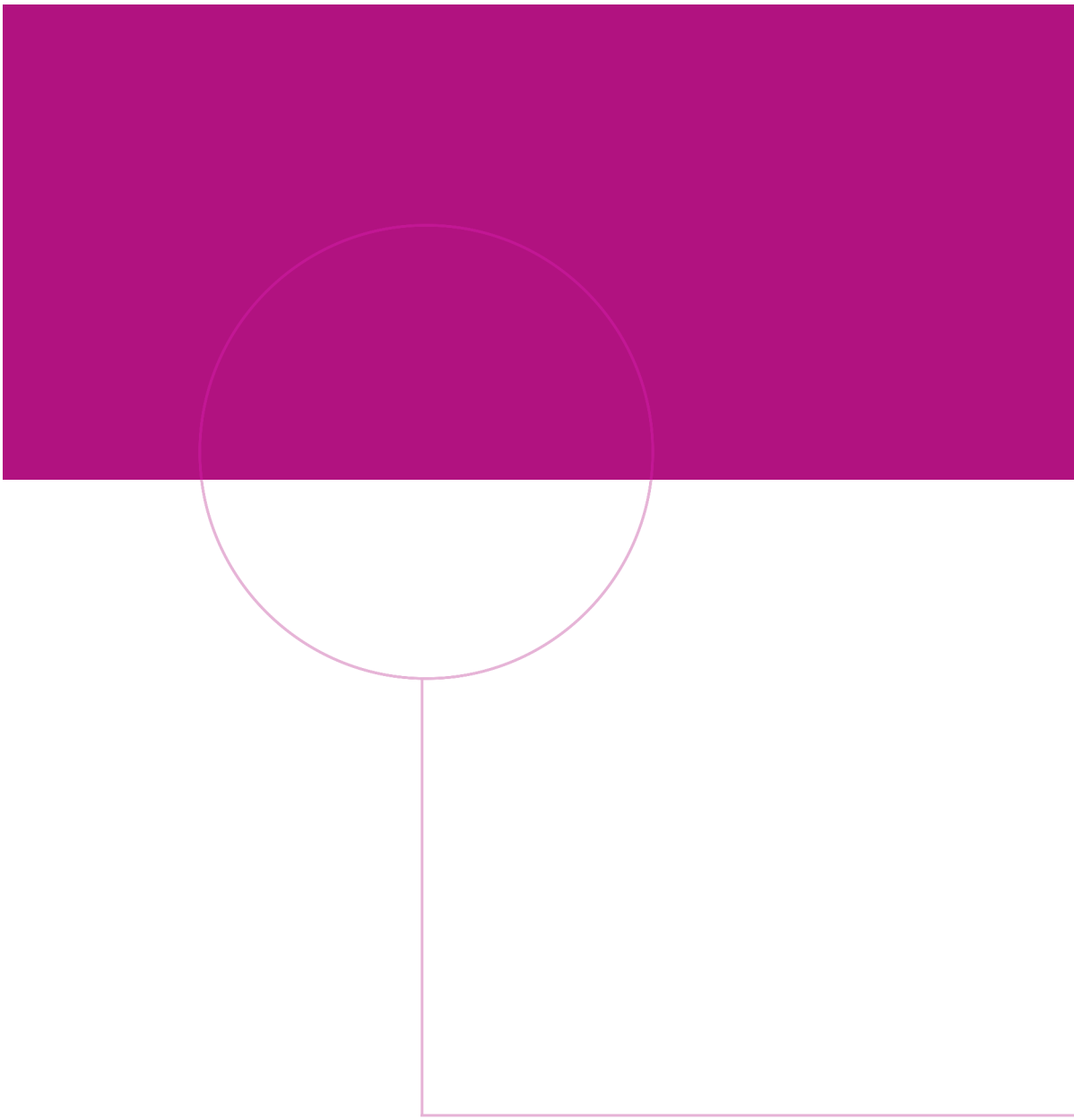
Bruker	**Svar**
Aleksander	Søkemenyen var bra. Dokumentasjonen var jo bra.
Peder	Veldig enkelt å finne fram til scriptene og dokumentasjonen han trengte. Ikke noe problem.
Anonym	Docs-nettsiden var han ikke så imponert av. Mareritt-nettside fra 2002. Et funksjonelt oppsett skulle vært vist med bilder i README. Masse eksempler på all funksjonalitet er praktisk når det kommer til Unity. Det er kronglete å finne fram med bare menyer.
Matilde	Scriptene var veldig greit. Litt usikker på manager/sensor. Veldig kjent for hennes del. God docs. Godt dokumentert kode.
Trym	Også ganske greit, bare har ikke gjort det på en stund. Vite layouten og hvordan pakken er lagd. Tydelig README, men kommenterer på en spesifikk setning han slet med.
Rinsai	Fant fram til skript. Etterspurte litt mer dokumentasjon. Ville egentlig finne closest entity i dokumentasjon.

Tabell 17: Brukeren opplevelse av filstrukturen og dokumentasjonen

Spørsmål 3: Hvordan var helhetsopplevelsen av pakken?

Bruker	**Svar**
Aleksander	Følte ikke han hadde fått nok innsikt i pakken til å ha noe helhetsinntrykk.
Peder	Veldig ryddig og bra laget. Fungerer "plug n play". Godt dokumentert kode. Bra at pakken inneholder samples, tests, osv. Inneholder meste han tenker man trenger.
Anonym	Virka gjennomført. Liker SpawnOnEntities eksempelet, var fint med et godt eksempel på bruk. Enkelt å implementere.
Matilde	Veldig grei. Fjernt fra ting som er vanlig for henne. Tok litt tid å sette seg inn i,men lett å få forståelse pga god dokumentasjon.
Trym	Veldig enkel å bruke. Enkel, forståelig. Føler det var veldig brukervennlig.
Rinsai	Bra. Ikke mye å si på det. Etterspurte dokumentasjon i Unity sin UI, og viste hvordan dette implementeres.

Tabell 18: Brukerens helhetsinntrykk av pakken



NTNU

Kunnskap for en bedre verden