# Diskussion

### Projektforløb

Projektet bærer præg af at der har været mange bolde i luften på samme tid og projektgruppens medlemmer har været udfordret både i analysen af behovet og udviklingen af prototypen. Der har været mange overvejelser igennem processen og beslutninger der skulle tages undervejs. Det har været nødvendigt at afgrænse projektet på alle punkter, da der har været mange inputs fra eksterne fagpersoner om hvilken retning projektet kunne tage.

### Mini-MTV

Det kan diskuteres hvorledes der kan sættes et estimat for hvor mange skridt og hvor meget tid personalet kan sparre med PatientCare, da det afhænger af mange faktorer som er uforudsigelige.   
Der er valgt at implementere PersonaleApp’en på en smartphone men det kan diskuteres om det kunne have været mere hensigtsmæssigt at personalet havde mulighed for at se patientkaldende ved at kigge ned på et display i stedet for at skulle tage en smartphone op ad lommen for at se hvad patientkaldet drejer sig om. Det er især at foretrække i situationer hvor personalet har hænderne fulde. Det kan ligeledes diskuteres om det er etisk korrekt at differentiere mellem tilbud til indlagte patienter ved at kun dem der har en smartphone kan benytte sig af dette supplement til kaldesnoren.   
Det skal også overvejes hvilken betydning det har for patienten at kaldet kun sendes til én primær personale. Under udviklingen af PatientCare blev denne løsning valgt på baggrund af den løsning gyn-obs har i dag. For at personalet bedre kan videredele patientkaldene talte gruppen om at patientkaldet med fordel kan sendes videre til en sekundær person eller det resterende personale efter et bestemt tidsinterval eller ved at den primære personale aktivt vælger at sende kaldet videre.

### Designprocessen

Designprocessen bærer præg af at projektgruppens medlemmer har haft forskellige tilgange til design af de forskellige moduler. Tilgangen til design af PatientApp har været meget direkte uden for mange overvejelser først, da udvikleren på forhånd havde et godt kendskab til iOS styresystemet. Tilgangen til design af PersonaleApp har været meget nyt, fordi det blev udviklet i Android, hvilket var forholdsvis nyt for projektgruppen. Designet er derfor blevet lavet ud fra et højt abstraktionsniveau. Tilgangen til design af AdminApp har også været at sætte abstraktionsniveauet højt men ud fra et andet perspektiv, der gjorde at fremtidige ændringer ikke ville være et problem.

### Push

PatientCare benytter sig af pull for hele tiden at holde data opdateret. Performancemæssigt vil det dog være en bedre løsning at gøre brug af push, fordi man i forhold til pull ikke forgæves brænder PatientAppen’s CPU resurser og tid af i en situation, hvor det er ikke nødvendigt at data opdateres – fx tjekkes der hele tiden på, om nyt patientkald er blevet forespurgt – i stedet vil det optimale være kun, at give besked til de moduler, der har behov for at vide det. Push funktionen kan implementeres i WebAPI’et som kan sørge for at skubbe patientkaldet videre til PersonaleApp. Men konceptet er prioriteret højere end performance i det tilfælde og derfor gøre PatientCare brug af pull.

### Overvejelser i forbindelse med en udvidelse af PatientCare

I forbindelse med en fremtidig udvidelse af systemet er der overvejet følgende scenarier:

* Den nærmeste sygeplejerske skal modtage kaldet og ikke andre (Lokalisering af nærmeste)
* Alle sygeplejersker på afdelingen skal modtage alle kald fra patienter
* Det skal være synligt for andre sygeplejerske at et kald er ”taget” af en kollega
* Der er knyttet primær og sekundær sygeplejersker på en patient og kun de to skal modtage patientens kald
* Det skal være muligt at ”tage” opgaver før de udføres
* Der er behov for at man kun klikker færrest mulige gange for at udføre en opgave
* Det skal være muligt for patienten at fortryde et kald der er sendt afsted
* Det skal være muligt for sygeplejersken at oprette et kald for en patient via sygeplejerskens arbejdstelefon
* En sygeplejerske skal kunne ”tage” flere opgaver inden hun udfører dem

En af de mere væsentlige overvejelser med hensyn til en udvidelse er lokalisering af patient. Som nævnt i afgrænsningsafsnittet er det et problem, at patienten har mulighed for at oprette et patientkald uafhængig af, hvor patienten befinder sig. Det er især et problem hvis patienten ikke er indlagt på et sengeafsnit. Columna Service Logistics benytter bl.a. en teknologi ifm. at spore en sengs flow i systemet, hvilket er til gavn for portører, der skal transportere senge. Denne teknologi er RFID og lignende løsning vil kunne tages i bruges til PatientCare til at tracke patienternes smartphone og dermed sikre et mere pålideligt system. Alternativt kunne GPS og Wifi benyttes men det er ikke præcist nok på et hospital med mange etager.

### Patientens rettigheder

Patientens rettigheder er minimale. Det er op til den enkelte afdeling hvilke valgmuligheder patienten skal have for at sende et patientkald afsted gennem PatientCare. Som udgangspunkt er det ønskværdigt at patienterne ikke skal kunne have mulighed for at modificere på det valgte kald. Det er derfor forudbestemt, hvad der er af valgmuligheder til at kunne give en årsag til at hidkalde personalet. Det betyder, at patienten som udgangspunkt ikke kan tilføje kommentar til det valgte kald.

### Github

Dem som stod for udviklingen af PesonaleApp havde ingen erfaring med Android Studio, og derfor var der en del opstarts problemer, fordi det var forsøgt at benytte versionsstyring direkte i Android Studio sammen med Github. Et problem var f.eks. når to fra projektgruppen skulle commite hver af deres opgave, som de stod får, hvilket skulle merges - og ikke altid har lykkes.

### En løsning uden Xamarin

Ud fra kravspecifikationen skal PatientApp’en udvikles i et miljø der gør det muligt at understøtte de tre største smartphone styresystemer iOS, Android og Windows, da PatientApp’en er tiltænkt til at være en *Bring your own device* løsning. Den traditionelle måde at løse det på, er at udvikle en app for sin platform i hvert sit sprog.

De tre sprog er:

* Objective-c eller swift
* Java
* C#

Dette vil kunne opfylde behovet for at udvikle PatientApp’en til de 3 platforme, men kræver at man kan udvikle i de tre sprog. Designmæssigt skal der designes en backend og en UI for hver platform, hvilket vil tage tre gange så lang tid at udvikle.

### Xamarin

En af de udfordringer der har været med at udvikle PatientApp’en som en cross-platform løsning har været tilgangen til Xamarin platformen og opsætte et cross-platform projekt med Visual Studio. Det har været en udfordring, fordi det er ny teknologi, og det kræver meget indsigt i forskellige styresystemer og deres API’er på trods af, at applikationerne programmeres i C#. Det har især været en udfordring at få bygget og deployet et Xamarin.iOS projekt i Visual Studio. Da Visual Studio er en IDE, der som udgangspunkt kun kan køres på Windows, kræver det en virtuel maskine på en Mac OSX computer, hvor en Mac Build OS server på Windows skal kunne snakke med en Mac Build OS server på Mac OSX. Kan man ikke kobles på build serveren i Visual Studio, kan projektet ikke bygges. Og det har haft udfordringer, fordi versionen af denne server skal passe i begge styresystemer – både i den virtuelle og ikke-virtuelle. Og da Xcode midt i perioden kom med en ny version, skulle build serveren opdateres, hvilket gav nogle problemer i forhold til at holde den synkroniseret med den rigtige version på Mac OSX og Windows. En anden udfordring har også været at bygge PCL’en i hvert projekt. Sker der en ændring i PCL’en, skal den bygges for hvert projekt, hvor den er inkluderet, hvilket man som udvikler godt kan glemme. Og det har betydet at ændringen ikke har berørt de projekter, man ønskede.

#### Xamarin.Forms

I forbindelse med teknologiundersøgelse mht. Xamarin stødte projektgruppen ind i nogle bedre tekniske alternativer efter et besøg hos Mjølner Informatics, der afholdt et oplæg omkring Xamarin. Her blev der snakket om to Xamarin løsninger. Den ene løsning var den traditionelle løsning, som PatientCare har valgt, og en anden løsning som gør brug af Xamarin.Forms. På dette tidspunkt var projektgruppen allerede godt i gang med implementeringen af den traditionelle løsning men Xamarin.Forms ville have været mere effektiv og ville have være hurtigere for ”Proof of Concept” af PatientCare. I stedet for at hvert view og dens kontroller skal programmeres for hver platform, vil man med Xamarin.Forms kun skulle gøre dette én gang. Fx vil et tekstfelt med blive til et UITextView på iOS, EditText på Android og TextBox på Windows.

**Fordele**

Xamarin.Forms bygger også på princippet om at have en C#, fælles kodebase som den traditionelle Xamarin løsning. Men med Forms deles brugergrænsefladen også og dermed deles koden 100 %.

**Ulemper**

Brugergrænsefladen skal skrives direkte i C# eller i XAML, da det ikke er muligt at bruge drag-and-drop funktionalitet som med den tradionelle løsning, eftersom brugergrænsefladen er forskellig. Derudover er custom rendering svært og bør undgås i Xamarin.Forms, da man ellers ligeså godt kan bruge den tradionelle Xamarin løsning. Styling af layoutet er også svær, da hver layout er forskellig fra hver platform, fx kan font størrelsen være forskellige selvom samme størrelse er sat for dem alle. Derudover skal man kende til de forskellige platformes API’er på trods af alt skrives i C#.

For PatientCare systemet har den tradionelle Xamarin være en god løsning. Men havde systemet benyttet sig af Xamarin.Forms vil det udviklings-og tidsmæssigt havde været en bedre løsning for ”Proof of Concept” sammenlignet med den nuværende løsning. Prototypen af PatientApp ville dermed også havde nået en Android og Windows løsning. Det ville have betydet, at PatientCare ville have levet op til konceptet om ”bring your own device”, som var det mål, der blev sat fra start.

### Android

#### Brug af service

PatientCare systemet benytter sig kun af pull requests og det er ikke særlig hensigtsmæssigt over for personalet. Det gør nemlig at personalet, manuelt, bliver nødt til at opdaterer listen med afventende kald. Det bedste vil være at, hver gang der blev oprettet et nyt kald ind i systemet, skal dette kald blive sendt til personaleApp’en og dermed gøre det til push beskeder i stedet for pull.

Det kræver brugen af en teknologi, som gør det muligt at lave disse push beskeder, men er ikke noget projektgruppen har valgt at ligge vægt på til udførelse af prototypen. Prototypen er blevet implementeret med en Android service som står for at lave pull requests hele tiden. Dette gør at personalet ikke manuelt, skal opdatere listen med afventende kald, men da det er gentagende pull requests er der også muligheden for at der ikke er kommet nogle nye kald og er dermed med til at skabe dårlig performance. Det ønskes at denne funktionalitet bliver noget af det første der skal laves i en evt. videreudvikling af systemet. Det er valgt at bruge en service fordi, da en service kan håndtere baggrundsopgaver, selv om app’en ikke er aktiv. Alternativt vil man kunne have brugt en baggrundstråd til at løse samme problem, men en baggrundstråd er kun aktiv så længe app’en er aktiv. Det vil sige at hvis personalet lagde telefonen i lommen og låste telefonen, så ville app’en ikke være aktiv og baggrundstråden vil heller ikke være aktiv. Derfor har der valgt en service frem for baggrundstråd. Men som tidligere nævnt er dette en nød løsning i forhold til den meste optimale løsning. For at se hvordan servicen er konstrueret, og hvordan PersonaleApp’en benytter den i praksis, henvises der til underafsnittet ”Bound Service (Modtag kald) under designafsnittet for Designdokumentet.

### WebAPI

Et af de helt store problemer for WebAPI’et har været skiftet fra at bruge *Linq to SQL* klasser til at bruge MongoDB frameworket for at tilgå databasen. Implementeringen for dette var ukendt for gruppen og krævede en del tid at lære. Det resulterede i at nogle beslutninger blevet taget for derefter at have fundet frem til en bedre løsning. Fx når alle aktive patientkald hentes fra databasen, bliver de leveret som en lang json formatteret tekststreg, hvilket er svære at omdanne til et objekt i koden. I stedet ville en bedre løsning være at benytte sig af json objekter, fordi det gøre det nemmere at omdanne – og er også mere læseligt.

### AdminApp

Det skal ikke være muligt for enhver at logge sig ind på administrationsmodulet og derved kunne slette eller oprette nye valgmuligheder. Det skal kun være administratorer af PatientCare systemet der kan tilpasse systemet. Men på nuværende tidspunkt er der ikke udviklet sikkerhedsmæssige anordninger for dette.

Et andet problem er at AdminApp forsøger at benytte MongoDB på en måde som databasen ikke er bygget til. Da databasen er en MongoDB og ikke har relationer mellem de forskellige collections, og da vi har valgt at opbygge databasen efter den relationelle tankegang, gør det bl.a. at en kategori på en type ligger som et id og ikke navnet på kategorien. Det gør at AdminApp bliver nødt til at hente informationerne fra kategori collectionen på MongoDB ud, inden de kan vises på siden. Hvis en kategori bliver slettet, er der dermed ikke et id som hører til det id, som en type har som sin kategori. Programmet vil derfor fejle, fordi det ikke kan hente den efterspurgte data, eftersom det ikke findes mere. I en relationel database, vil man kunne lave *On Cascade delete* på en foregin key relation, men i MongoDB, skal dette gøres manuelt og det er ikke blevet implementeret på WebAPI’et.

### Databaseskift

Det blev besluttet at bruge en relationel database og lagre data i tabeller med kolonner og rækker. Beslutningen om at gøre brug af en relationel database blev taget ud fra det faktum at alle i gruppen har benyttet denne form for database og fordi datastrukturen til applikationerne i PatientCare passer godt til den relationelle datastruktur.

Projektgruppen fik en studielicens til Microsoft Azure som blandt andet tilbyder brugen af MySQL databaser på en databaseserver. Her oprettede gruppen tabeller i overensstemmelse med den datastruktur som var fastlagt for PatientCare. På grund af et begrænset antal *credits* som hurtigt blev opbrugt på Microsoft Azure, var gruppen nød til at tage en beslutning om at skifte database. Det ville ikke være muligt at lave en ny studielicens til Microsoft Azure, da samme problem ville opstå igen. Ingeniørhøjskolen Aarhus kunne tilbyde at bruge deres database, men da den er afhængig af VPN (Virtual Private Network) er det ikke muligt for vores projekt. Projektgruppen valgte i stedet at bruge MongoDB som systemets fælles database. Tankegangen med datastrukturen er ikke den samme i MongoDB som for relationelle databaser og det betød at datamodellen for systemets var nødt til at blive ændret lidt. Værst gik det ud over WebAPI’et hvor et helt nyt framework og en hel ny database struktur skulle benyttes.



WebAPI’et benytter et repository pattern, hvilket ideelt set gør at databasen kan udskiftes uden det har betydning for systemet. Ved et eventuelt databaseskift, vil man ikke skulle ændre noget i den eksisterende kode men lave en ny implementation af sit repository pattern så systemet kan tilgå den nye database.

Databasen forbindes til resten af systemet gennem WebAPI’et og da WebAPI’et er et webinterface til resten af systemet spiller det ingen rolle for vores applikationer, hvilken database der ligger bag ved. WebAPI’et modtager og sender data i json format til og fra vores applikationer. Det har ingen betydning for databasen, da omdannelsen af json format til det format som databasen bruger, sker i WebAPI’et.

### Systemsikkerhed

En af de grunde til at tanken om integration til Systematic havde fyldt så meget i de indledende faser af projektet, er bl.a. at et stort IT hus som Systematic, er et IT hus man kender og inde for healthcare sektoren og er et firma man stoler på. Og her kommer det aspekt om systemsikkerhed ind i billedet. Hvis vi havde haft muligheden for at implementere PatientCare’s Patientkald ind i Systematics eksisterende opgavesystem, ville det problem være løst allerede. Systematic’s systemer bliver allerede brugt på nogle hospitaler i dag og derfor har Systematic allerede været tvunget til at tænke systemsikkerhed, og det vil derfor ikke være noget PatientCare systemet skulle bekymre sig om at løse.

Hvorfor er der ikke lavet patientkald med årsag førhen? Teknologierne findes til det. Hvilke udfordringer er der med at implementere på offentlige hospitaler?

### WebAPI

I den videre udvikling af WebAPI’et er der umiddelbart tænkt over følgende punkter,

* Sikkerhed, for patientApp når patient logger ind, skal der laves en validering af om patienten er indlagt og om patienten må oprette patientkald i systemet. For personaleApp skal logges ind med brugernavn og password som gemmes et sted i databasen og valideres når personalet logger ind.

Umiddelbart vil det kunne løses ved at tilføje brugernavn, salt og password felt på et personale, hvor salt bliver brugt til at finde frem til det krypterede password. Det kunne også tænkes at der skulle interageres med et AD hvor personalet er oprettet i, men det skal der tages stilling til på et senere tidspunkt.

* Tildelt personale, når et patientkald bliver sendt ind i systemet, skal der kigges på hvilket personale der er primært og sendes til det primære personale først, hvor der efter et tids interval bliver sendt videre til andet personale, hvis patientkaldet ikke blev udført af det primære personale.
* Generelt kæves der en anden form for teknologi end det der er lavet på nuværende tidspunkt, men burde kunne løses ved at implementere SignalR teknologien. Kort sagt vil det gøre os i stand til lave push beskeder til patientApp og personaleApp i stedet for at skulle lave pull requests hver gang data skal opdateres. Uden at vide om det skal implementeres i WebAPI’et, er det taget med i disse overvejelser.

**Validering af patient**  
På nuværende tidspunkt validerer systemet patienten ved at sammenligne det CPR-nummer der bliver sendt afsted med det nyoprettede kald med de CPR-numre der er lagret i fællesdatabasen. Hvis det stemmer overens er det udtryk for at patienten med dette CPR-nummer er indlagt. Fremsigtet er det meningen at der skal laves en bedre validering af om patienten er indlagt eller ej. Det er ønskværdigt at patientens indlæggelsestidspunkt benyttes af sikkerhedsmæssige årsager for at tjekke om patienten er indlagt og på den måde sikre at kun indlagte patienter kan sende patientkald afsted. Dette kan foregå via WebAPI’et ved at tjekke i fælles databasen ud fra data mock om patienten har et indlæggelsestidspunktet der er før ”nu” og et udskrivelsestidspunkt der er efter ”nu”.



På diagrammet ovenfor ses et aktivitetsdiagram over de overvejelser projektgruppen har gjort sig I forbindelse med validering af om patienten er indlagt eller ej.

### AdminApp

Foruden en bedre implementeringen af de omtalte problemer, skal AdminApp fremadrettet videre udvikles til at kunne håndtere flere afdelinger på et sygehus og det skal være muligt at tilpasse de forskellige afdelinger uafhængigt af hinanden. Det gælder for alle PatientCare’s apps at det vil kræve løbende kontakt til patienterne og personalet på hospitalerne for at tilpasse systemerne. Det er fejlagtigt og tro at man kan fast sætte en kravspecifikation og så udvikle på baggrund af denne og ikke inddrage brugerne af systemerne.