Indholdsfortegnelse

SCRUM 2

Vandfaldsmodellen 2

Unified Process 2

Spiralmodellen 3

XML 3

Java 4

SQL 4

Use cases 5

Brugerscenarier 5

Accepttest 5

Brugertest 5

Entity Relationship diagram 5

Attribut 5

Use case diagrammer 6

Klassediagrammer 6

Sekvensdiagrammer 6

Aktivitetsdiagrammer 6

Domænemodellen 6

Bilag 7

SCRUM er **agil** og er en udviklingsmetode som har meget fokus på udviklingen af software. Her ser man det som en kompleksitet og en uforudsigelige fremgangs proces, som gør at den er langt fra en planlagt proces. Dvs. at man ikke nødvendigvis har tilrettelagt sig en proces, som gør at man støder i følgende problemer. Dette er f.eks. at man ikke kender alle **kravspecifikationer** i starten af processen, kravene kan ændre sig løbende og at alt som der udføres kan være uforudsigelig, da ny teknologi eller værktøjer kan komme på spil. Alt i alt betyder det at processen kan starte med hvilken som helst aktivitet, og skifte sig til en ny på ethvert tidspunkt. I **scrum** har man roller som er essentielle for hele processen, samt **artefakter**.

Rollerne er **product owner**, **scrum master, development team. Product owner** har ansvaret for at planlægge, således at produktet leveres til klienten via en effektiv **kravstyring**. **Scrum master** sikrer sig at gennemførelsen af de forskellige sprints, og holder en forbindelse og klargøre processen for product owner. **Development team** står for at sikre sig og færdiggøre de enkelte **sprints**, samtidig med at de fastholdes indenfor den fastlagte budget. **Artefakterne** er **produkt backlog**, **sprint backlog**, **sprint**. I **produkt backlog** ser man på alle forskellige krav til systemet, som håndteres af selve ejeren. Her fokuserer man på vigtigheden af kravet. **Sprint backlog** er hvor man ser på en **scrum-gruppe**, som skal implementere den kommende **sprint** i processen. Den sidste er **sprint**. Her inddeler man arbejdet i **sprints**, som max varer i 30 dage. De forskellige **sprints** inddeles til møder (**sprint planning**), hvorpå man afslutter med at fremvise en ny version (**sprint review**).

Vandfaldsmodellen er en model hvor man følger forskellige trin i forhold til hvad, der skal prioriteres først i processen. En følger man specifikke faser i en bestemt rækkefølge, som er: **kravspecifikation**, **design**, **kodning**, **integration**, **afprøvning og fejlfinding**, **installation**, **vedligeholdelse**. Det der er specielt ved fasen er, at den er **sekventiel**. Det vil sige at man ikke hopper i **faserne**, men at man strengt forhold til den **fase** man er i, og ikke springer tilbage. Så hvis man mener at man er færdig med **kravspecifikation**, så går man videre til næste trin, som er **design**. Fordelene ved **vandfaldsmodellen** er, at det som man leverer til kunden i sidste ende er **kvalitetssikret** samt er den også nem og overskuelig og sparer derfor tid. **Ulemperne** er, at hvis man laver en **fejl** i et af de første trin, så kan det skabe problemer senere henne i projektet, som fører til **øget omkostninger**. Modellen er ikke så god til og håndtere forandringer. Dvs. at når man først har lagt **kravspecifikationer**, så kan det være et problem at ændre det længere nede i processen.

Unified Process kan forkortes som **UP**, og er en **iterativ systemudviklingsmetode** som består af 4 overordnede faser, **forberedelse**, **etablering**, **konstruktion**, **overdragelse.** I f**orberedelse** ser man på ens overblik over de krav **klienten** har lagt til grunde overfor systemet. I næste fase, **etablering**, ser man på de systemets centrale dele og skaber sig en forståelse af kravene og arkitekturen.
I **konstruktion** fokuserer man på at **teste** samt **udvikle** de funktioner systemet skal kunne, og som i sidste ende fuldføre kunden og slutbrugernes tilfredsstillelse. I sidste fase som er **overdragelse**, også kendt som **afslutningsfasen**, slutgøre man systemet og levere det til brugerne. Udover det, ser man også på problemer, fejl og mangler ved systemet.

Spiralmodellen er en kombination af **vandfaldsmodellen** og den **iterative** model. Hver fase i **spiralmodellen** begynder med en design mål og ender med at klienten reviewer modellen. Udviklingsholdet i **spiralmodellen** starter med at lave en lille sæt af krav og går igennem en **udviklingsfase** for de her krav. **Udviklingsholdet** tilføjer funktionalitet til kravene hvor **spiralen** sammen med kravene vokser indtil man når til **produktionsfasen**.

Planlægning hvor man estimerer **prisen**, **skemaet** og **ressourcerne** for **iterationen**. Den involverer også forståelsen for **systemkrav** og kommunikationen mellem **systemanalytiker** og **kunden**.
Man bruger **risk analysis** til at identificerer den **potentielle risici** og den **risikoreducerende strategi** er planlagt og færdiggjort. **Ingeniørarbejde** der inkluderer test, kodning og implementering af produktet. **Evalueringsdelen** er hvor kunden evaluerer produktet og det inkluderer også overvågning af ekstraomkostninger og overarbejde. En spiral model er bedst at bruge når projektet er stort. Nogle af fordele ved spiral modellen er at man kan tilføje flere krav og ændringer senere i projektet. Af ulemper er der f.eks. risici for at komme uden for skemaet og budget. Spiralmodellen er ikke god til små projekter da det vil blive for stort i omkostninger.

Extensible Markup Language (XML) er til lagring og transportering af **struktureret tekst**.
XML filer skal have et rod element som definerer xml **version** og **tegnkodning**, hvilket kunne se således ud: <?xml version=”1.0” encoding=”UTF-8”?>
Elementer skal have åbnings og lukke **tags** i form af: <Element></Element>
XML gør forskel på store og små bogstaver, og **tags** skal derfor være identistiske.
Værdierne i attributter skal være i anførselstegn som set i rodelementet: ”UTF-8”
XML namespaces er attributter på elementer med en **Uniform Resource Identifier (URI)** værdi, som definerer ordbogen der bruges til elementet <element xmlns=”namespaceURI”>

XML skemaer bruges til at beskrive strukturen og **validere** syntax i et XML dokument. Skemaet er en del af XML dokumentet, og beskriver tilladte elementer og attributter, nummer og ordre af underelementer, data typer af elementer og attributter. **Se bilag 1 for eksempel på skema.**

Java XML Binding API (JAXB) bruges til at transformere Java kode til XML og tilbage igen. Java klasser kan laves om til XML skemaer og omvendt. Java klasser kan også kortlægges med annotationer som binder dem til XML. **Se bilag 2 for eksempel på transformering.**

Java er et objektorienteret programmeringssprog som består af et bibliotek af klasser. Interaktion mellem klasserne skal simulere interaktionen mellem det de repræsenterer i virkeligheden, f.eks. personer. Java kan køre på mange forskellige styresystemer, fordi java programmer kan køre i en **Java Virtual Machine (JVM)**. Dette gør også java langsommere end programmer skrevet til et specifikt styresystem, men det er attraktivt fordi man kan skrive ét program, og køre det på flere styresystemer.

Objekterne i java er det som karakterisere en **klasse**, f.eks. højde for klassen personer. Der findes **primitive typer** som bruges ved tildeling af en værdi til en variabel. **Int** bruges om **heltal**, **double** om **kommatal**, **String** om **tekst**, og **boolean** om noget **binært** (0/1, ja/nej). **Referencer** til klasser er en form for navngivning af objekter, f.eks Person p = new Person().

Der findes også **statements** som kan være **conditionals** eller **loops**. En conditional kan være *if* som skal opfyldes før der køres noget *then*. **En loop** kan bruges til at oprette **referencer** for **en klasse** og tildele dem **værdier**, ved at f.eks. køre igennem en database. **Statements** køres normalt fra **toppen** af en java fil **til bunden**, men **control flow statements** begrænser kørslen af kode ved brug af **conditionals**. Det kan være med statements som *if-then*, *if-then-else*, *while*.

Structured Query Language (SQL) er et sprog til lagring, manipulation og hentning af data i en database. SQLite er et relationelt database system i et bibliotek skrevet i sproget C. SQLite er ikke en databasen af typen klient-server, fordi databasen er en del af selve programmet, og der er ikke brug for en database server eller klient-server kommunikation.

En relationel database består af en tabel, hvor relaterede data har den samme nøgle. Eksempelvis kan forskellige data for den samme person være relaterede i en tabel over forskellige personer. Databasen manipuleres med sprogene **Data Definition Language (DDL)** og **Data Manipulation Language (DML)**. DDL bruges til oprettelsen af objekter i form af tabeller i databaser (CREATE TABLE) og bestemmer dermed formatet. DDL kan også ændre formatet senere, og fjerne objekterne fra databasen. DML bruges til at manipulere tabeller ved at indsætte, vælge, opdatere og slette data (INSERT INTO, UPDATE, DELETE). **Se bilag 3 for eksempler på SQL kode.**

Use cases er små tekst historier som man bruger til og opdage samt registrere de krav, som er nødvendige. Her ser man på hvordan en **aktør** bruger et system til og opnå et mål. En aktør kan være en person, computer system eller en organisation.
Brief er en sammenfatning, oftest omkring et **main success scenario**. **Casual** er et **uformelt** stykke format, som består af flere forskellige afsnit, som dækker forskellige scenarier. **Fully dressed** beskriver alle trin samt **variationer** skrevet i **detaljer**, som **preconditions** og **success guarantees**. I vores egen iteration har vi lavet en **casual use case** omkring "Registrering af vaccination", som består af use case name, scope, primary actors, pre-conditions, success guarantees, main success scenario, extensions og frequency of occurrence**. Se bilag 4 for eksempel på casual use case.**

Brugerscenarier opstilles med en indledning til programmets rolle, og hvordan programmet fuldføre de forskellige opgaver. Dette kan f.eks. være i form af, at man har en patient som skal indskrives på et hospital, hvor man så beskriver scenariet for hvordan primære aktørerne interagere med programmet, for og løse de daglige opgaver.

Accepttest ser man på de **systematiske processer**, og på en række ting som programmet skal kunne. Her ser man på hvordan mennesket udfører opgaverne ud fra ens produkt. I iteration 3 har vi lavet en accepttest, hvor dem som skal teste programmet skal oprette en borger, ud fra de informationer de har fået ved indledningen. **Se bilag 5 for eksempel på accepttest.**
Brugertest udføres i de sidste faser af ens softwareudvikling af selve programmet, og ser på programmets tilfredsstillelse. **Brugertests** er meget gode til og se hvor på hvordan den klarer sig i den virkelig verden. I vores iteration 3 har vi kigget på hvordan programmet klarer inden for hastighed, overskuelighed, positionering af knapper osv. og overordnet præstation, i forhold til brugerens behov. **Se bilag 6 for eksempel på brugertest.**

Entity Relationship diagram (ER diagram) er en data **modelleringsteknik** som illustrerer en systems **entiteter** og forholdet mellem **entiteterne**. En **ERD** kan bruges til at skabe en **database design**. Den kan bruges til at modellere data i **databasen** og til at skabe en **koncept design** af **databasen**. **En entitet** er en element der kan eksistere for sig selv i den virkelige verden. **Entiteter** er tilsvarende til tabeller i en **database**, i en **relationel database**, hvor hver række af **tabellen** repræsenterer en del af en **entity**.

Attribut af en **entitet** fortæller om den særlige egenskab der beskriver den **enhed**. **Relationer** beskriver interaktioner mellem **entiteter. Multiplicitet** forklarer antallet af gange en entitet kan eller skal være forbundet med antallet af en anden **entitet**. Generelt plejer **multipliciteten** at være **en-til-en**, **en-til-mange** eller **mange-til-mange**. Hvor **ER-diagrammet** er i sig selv et diagram, viser den strukturen af en **database**, mens de andre **diagrammer** som vi har set på i forhold til **UML** som er brugt til **objektorienteret programmeringssprog**. **Se bilag 8 for et eksempel på ERD.**

Use case diagram beskriver hvordan en **bruger** bruger et **system** for at komme frem til et **mål**. En use case bruges som en **modellering** som beskriver de funktioner der skal **implementeres** og nogle af de fejl man kan løbe ind i. Use cases består af: **Aktører** er den type af brugere som interagerer med **systemet**. **System** hvor use cases fortæller om de **funktionelle krav** som beskriver den måde en **system** forventes at bruges på. **Goals** er hvor use cases er startet af en **bruger** (**user**) for at opfylde **mål** som der beskriver de ting der skal gøres for at komme frem til et mål. **Use cases** er **modelleret** med brug af **UML** og er repræsenteret med **ovaler** der indeholder navnene på use case.
Aktører er repræsenteret som figurer som har navnet af **aktør**. Og de kasser der er rundt om use cases repræsenterer **systemet**. **Se bilag 11 for eksempel på use case diagrammer.**

Klassediagrammer er en type **diagram** og en del af **UML**, som definerer og giver et indblik i strukturen i et system i forhold til **klasser** deres **attributter, metoder og forholdene mellem dem. Klassen** definerer **attributterne** og **metoder** for mængde af **objekter**. Repræsenteres af rektangler med klassens **navn**. **Objekter** deler sammen "**opførsel**" og mængde af **attributter**. **Attribut** vises med **navn**, **type**, oprindelig **værdi** eller andre **egenskaber**. Vises også med synlighed. **+** står for åbne

(**public**) **attributter** og **–** står for **private attributer**. **Associationer er f**orhold mellem klasser. Fælles semantik og struktur, mange typer forbindelser. **Sammenligner til andre** resterende UML modeller. **Se bilag 9 for eksemplet på klassediagram.**

Sekvens/interaktionsdiagrammer beskriver forløbet i et **system** (eller **usecase**) over tid, samt hvilke **aktører** eller **objekter** der er i brug. I **Sekvensdiagrammer** viser udveksling af **meddelelser** mellem flere **objekter**, i en specifik, **tidsbegrænset** situation. **Sekvensdiagrammer** lægger særlig vægt på rækkefølgen og tiden når **meddelelserne** til objekter sendes. **Objekter** repræsenteres af **lodrette streger** i **sekvensdiagrammer**, med objektets navn øverst. **Tidsaksen er også lodret**, og vokser nedad, så meddelelser sendes fra et objekt til et andet i form af pile med operationer og parameternavn. **Se bilag 12 for eksempler på sekvensdiagrammer.**

Aktivitetsdiagrammer er en visualisation af et sæt af systems **aktiviteter**, **use cases** og et **systems funktioner** på et detaljeret plan. Et **aktivitetsdiagram** er repræsenteret af **former** der er forbundet med **pile**. **Pilene** går fra **start** af **aktivitet** og viser den **sekventielle rækkefølge** af **aktiviteterne**. **En sort cirkel** betyder start af **workflow**, **en cirkel rundt om den sorte cirkel betyder slut. Rektangler** betyder udført handling som er **beskrevet med tekst** inde i rektanglet. **En diamant** er brugt til at vise at der skal tages en **beslutning**. **Synkronisering Stænger** starter en **subflow** hvor der sker flere flere **handlinger** samtidig. **Se bilag 7 for aktivitetsdiagram**.

Domænemodellen er et **klassediagram**, der beskriver de væsentligste **objekter** i **domænet** og deres indbyrdes **relationer**. Man starter først med at identificere de **væsentligste klasser** og derefter tilføjer man den til **domænemodellen** hvor man tilføjer **attributter** og navngiver **relationer**. En **attribute** fortæller om den særlige **egenskab** der beskriver den **enhed**. **Relationer** beskriver **interaktioner** mellem klasser. Man kan bruge **nedarvning** hvis en **klasse** består af flere **klasser** som har forskellige **attributter**, f.eks. en sundhedsperson kan både hver læge og sygeplejerske men de har forskellige attributter. **Multiplicitet** forklarer i den her sammenhæng, antallet af gange en klasse kan eller skal være forbundet med antallet af en anden klasse. Generelt plejer **multipliciteten** at være **en-til-en**, **en-til-mange** eller **mange-til-mange**. **Se bilag 10 for eksempel på domænemodel.**

Bilag 1 XML skema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:Element name="Element">
        <xs:element name="Child" type="xs:string"/>
</xs:Element>
</xs:schema>
```

Bilag 2 Java XML Binding API (JAXB)

Java kode

```
@XmlRootElement
public class Element {
        public String Child;
}
```

XML kode

```
<Element>
        <Child>value</Child>
</Element>
```

Bilag 3 Eksempler på SQL kode

```
CREATE TABLE table_name (column_1 String, column_2 Int);
INSERT INTO table_name (column_1, column_2 VALUES (value_1, 'value_2');
DELETE FROM table_name WHERE some_column = some_value;
UPDATE table_name
SET some_column = some_value
WHERE some_column = some_value;
```

Bilag 4 Casual use case

Use case name	Registrering af vaccination
Scope	DDV
Primary actors	Praktiserende læge Lægesekretær
Pre-conditions	1. Server skal være aktiv således at personale kan logge på. 2. Felterne til udfyldning af registrering skal stå klart. 3. Skal være i stand til og gemme registreringer.
Success guarantees	Patienten er blevet registreret
Main success scenario	1. De er i stand til og logge på. 2. Personale kan finde vaccination. 3. Lægen kan påbegynde registreringen. 4. Registreringen bliver registreret i DDV og er synlig for personale.
Extensions	1a. fejl ved login af personale. 1. Systemet skal informere hvorfor fejl opstod. Dette er i form af forkert navn og kode eller hvis serveren er nede. 1b. kan ikke finde vaccination. 1. Fortæller hvor fejlen ligger i forhold til stavefejl osv.
Frequency of occurrence	Hver gang borgeren modtager en vaccination.

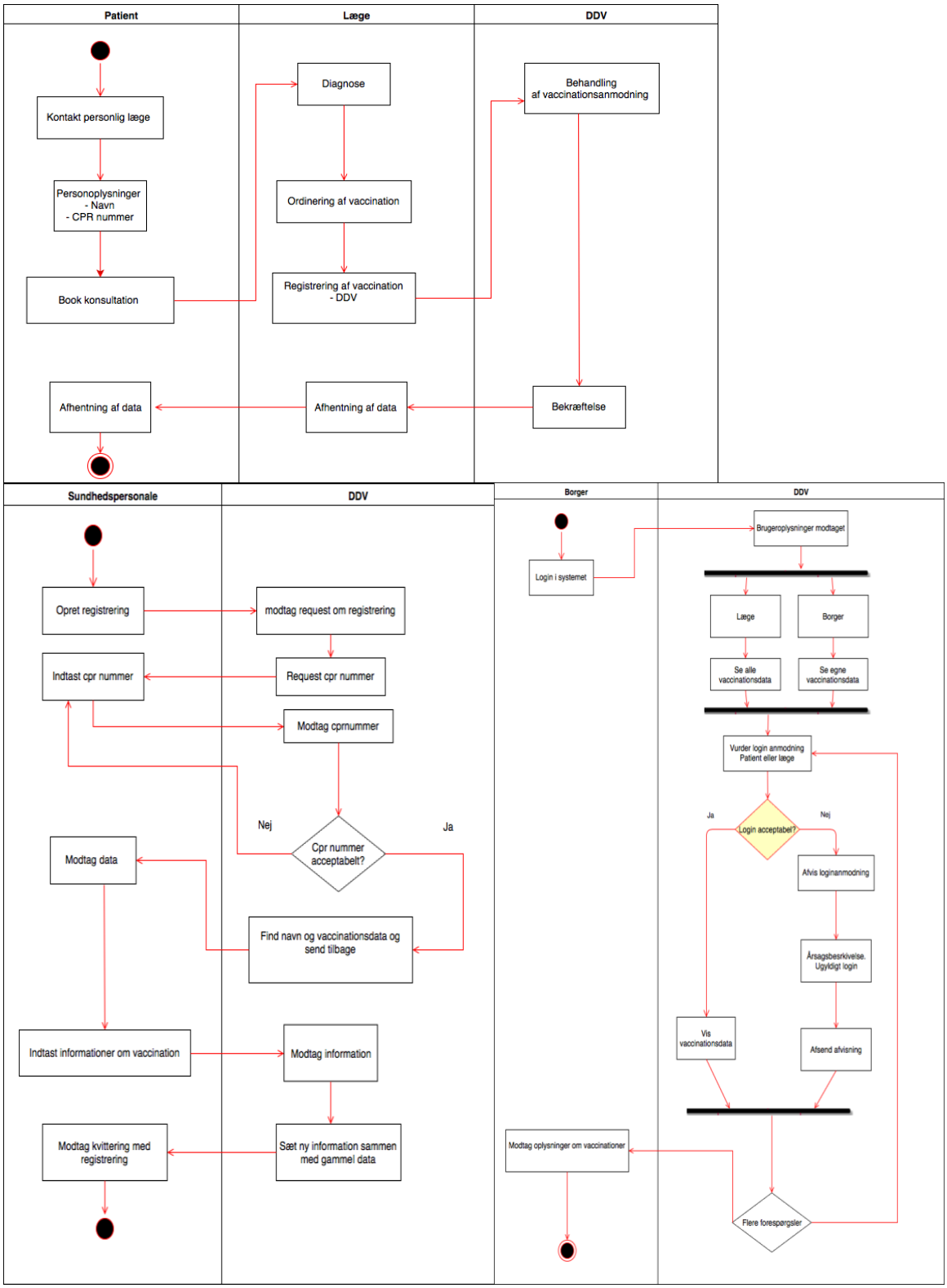
Bilag 5 Accepttest

Spørgsmål	**Svar (sæt ét kryds)**	
	Ja	Nej
Oprette en borger? Lykkedes det? (30 sek.)	**X**	
Hvis nej, hvad gik galt?		
Henter data fra databasen? Lykkedes det? (30 sek.)	**X**	
Hvis nej, hvad gik galt?		
Redigere i databasen? Lykkedes det? (40 sek.)	**X**	
Hvis nej, hvad gik galt?		
Gemmer ny-redigeret data? Lykkedes det? (60 sek.)	**X**	
Hvis nej, hvad gik galt?		

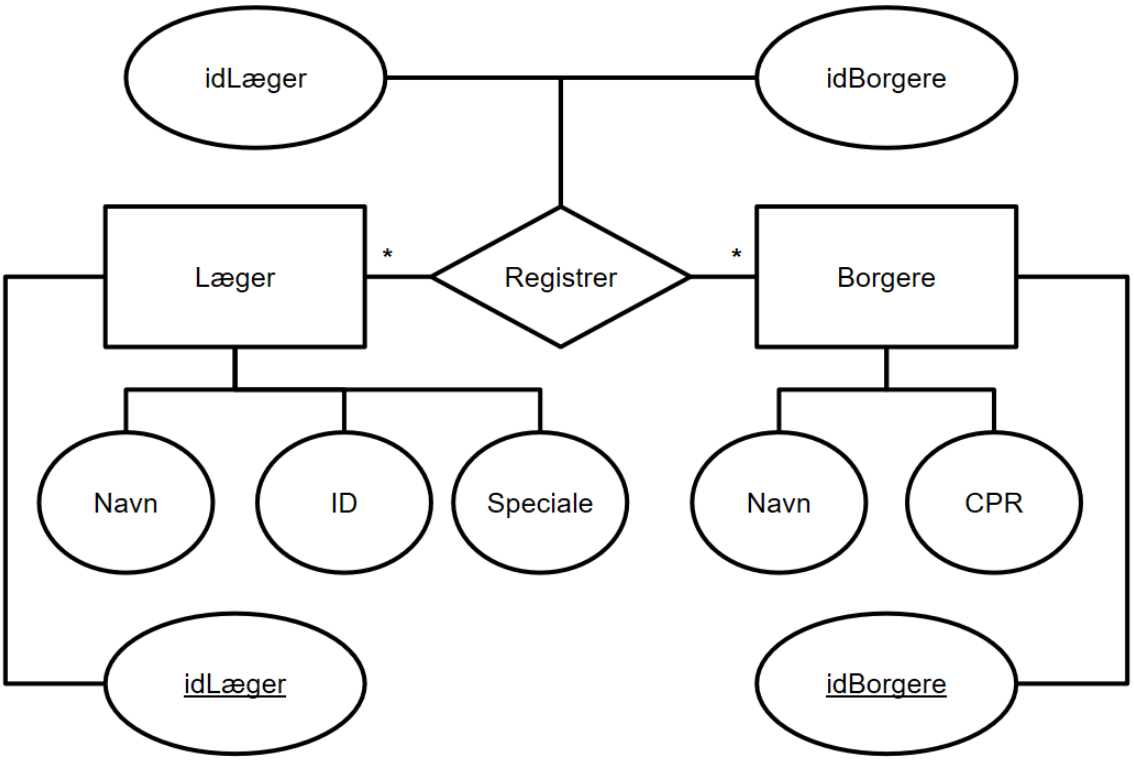
Bilag 6 Brugertest

Spørgsmål	**Svar (sæt ét kryds)**			
	Utilfreds	Mindre tilfreds	Tilfreds	Meget tilfreds
Hvor hurtigt hentes patienterne databasen?				**X**
Hvor overskueligt er designet?		**X**		
Fremkommer det tydeligt hvilke knapper man skal trykke?			**X**	
Hvad er din overordnede bedømmelse af programmet?			**X**	

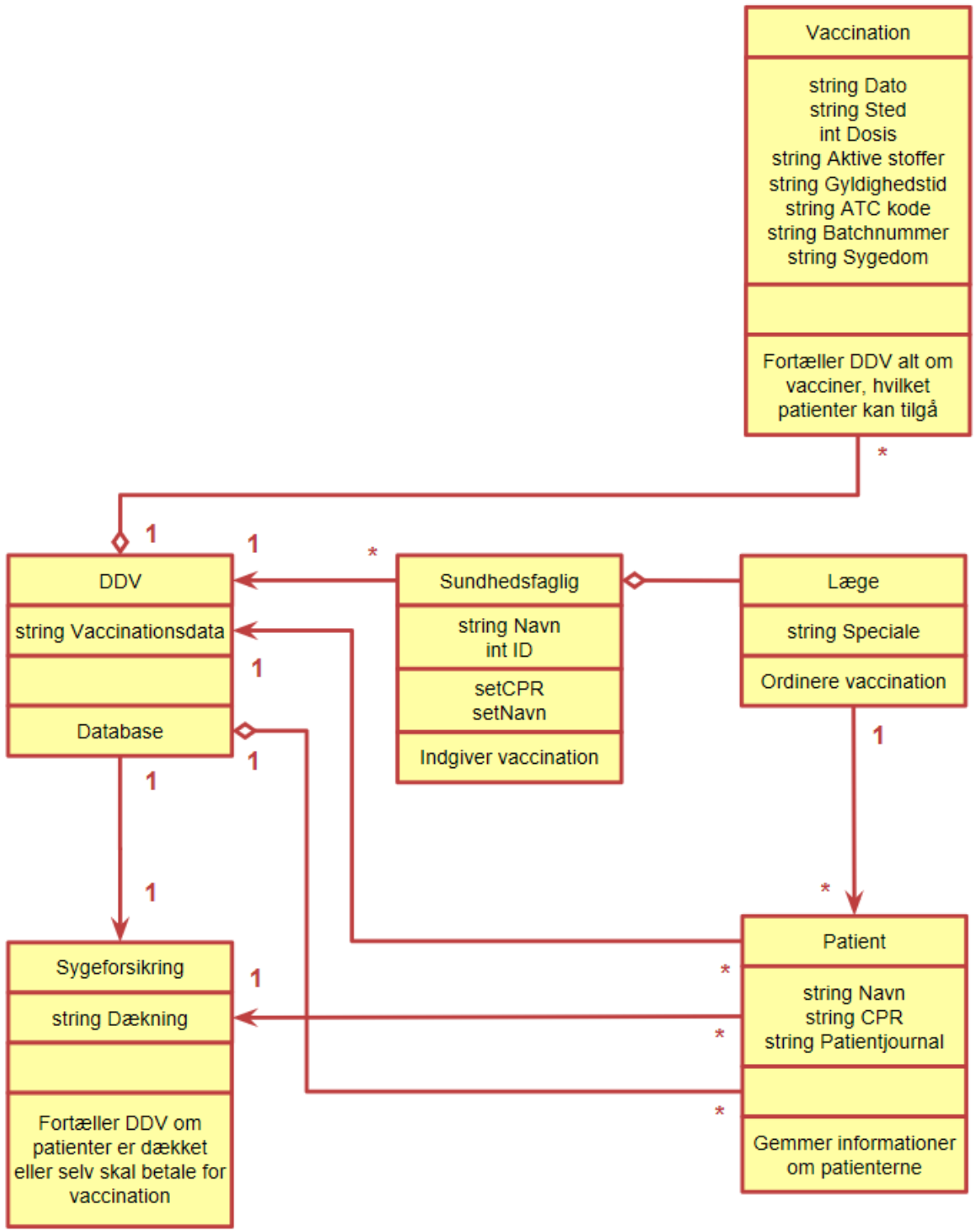
Bilag 7 Aktivitetsdiagrammer



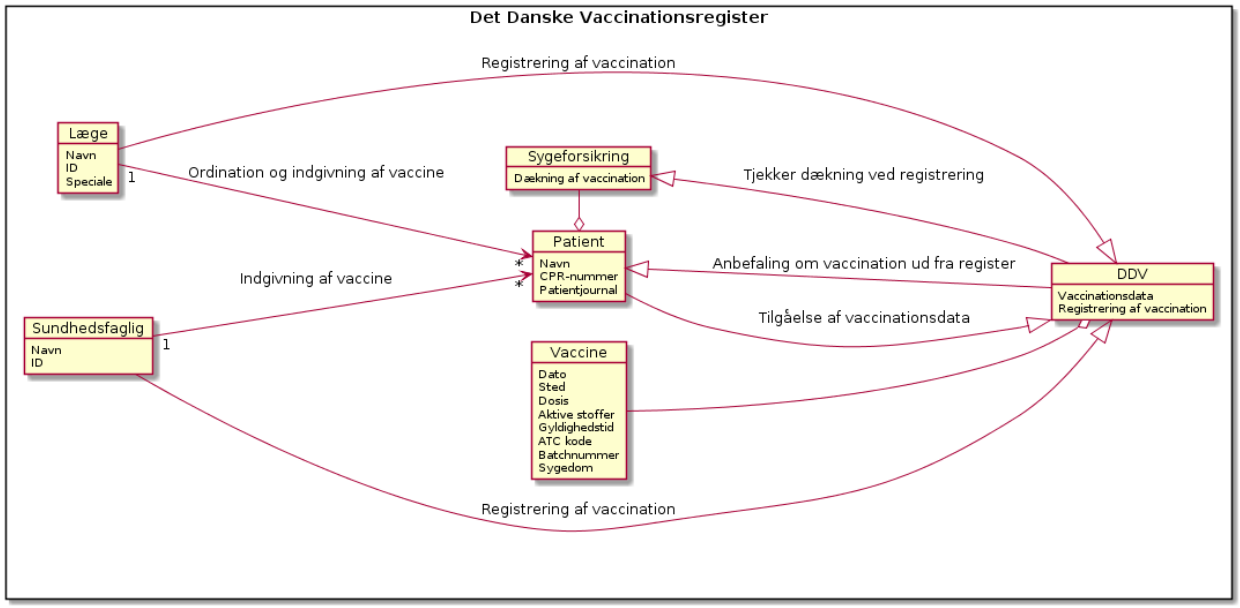
Bilag 8 ER-Diagram



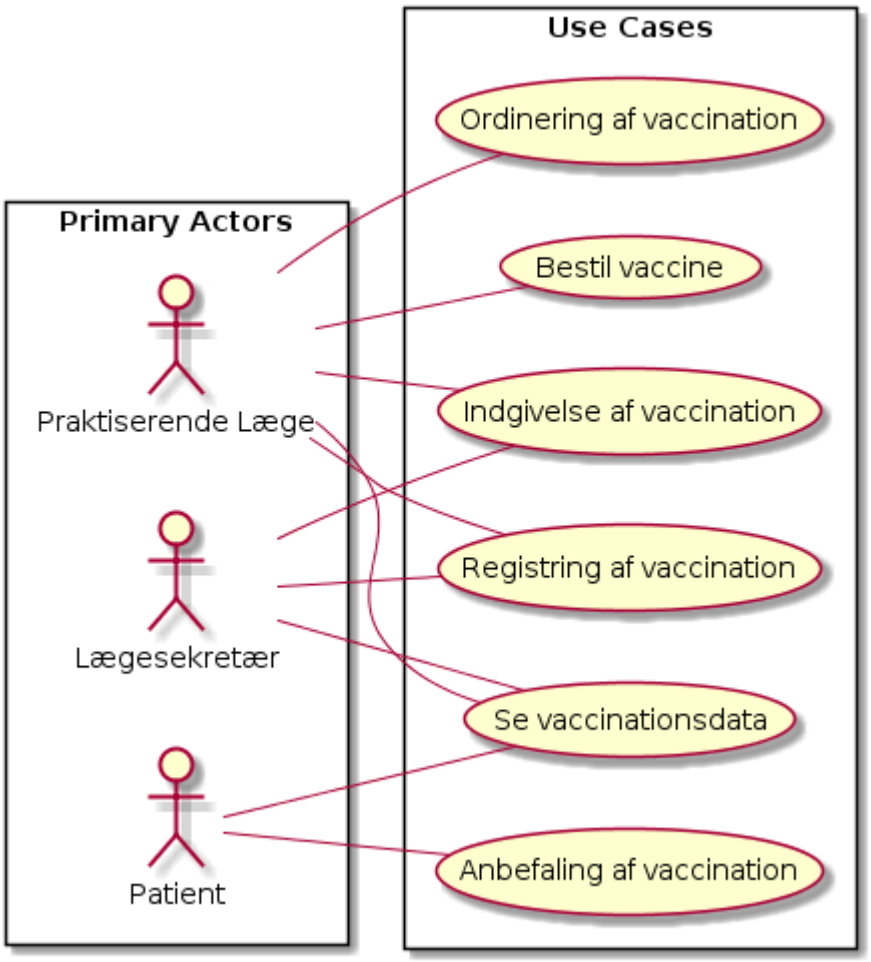
Bilag 9 Klassediagram



Bilag 10 Domænemodel

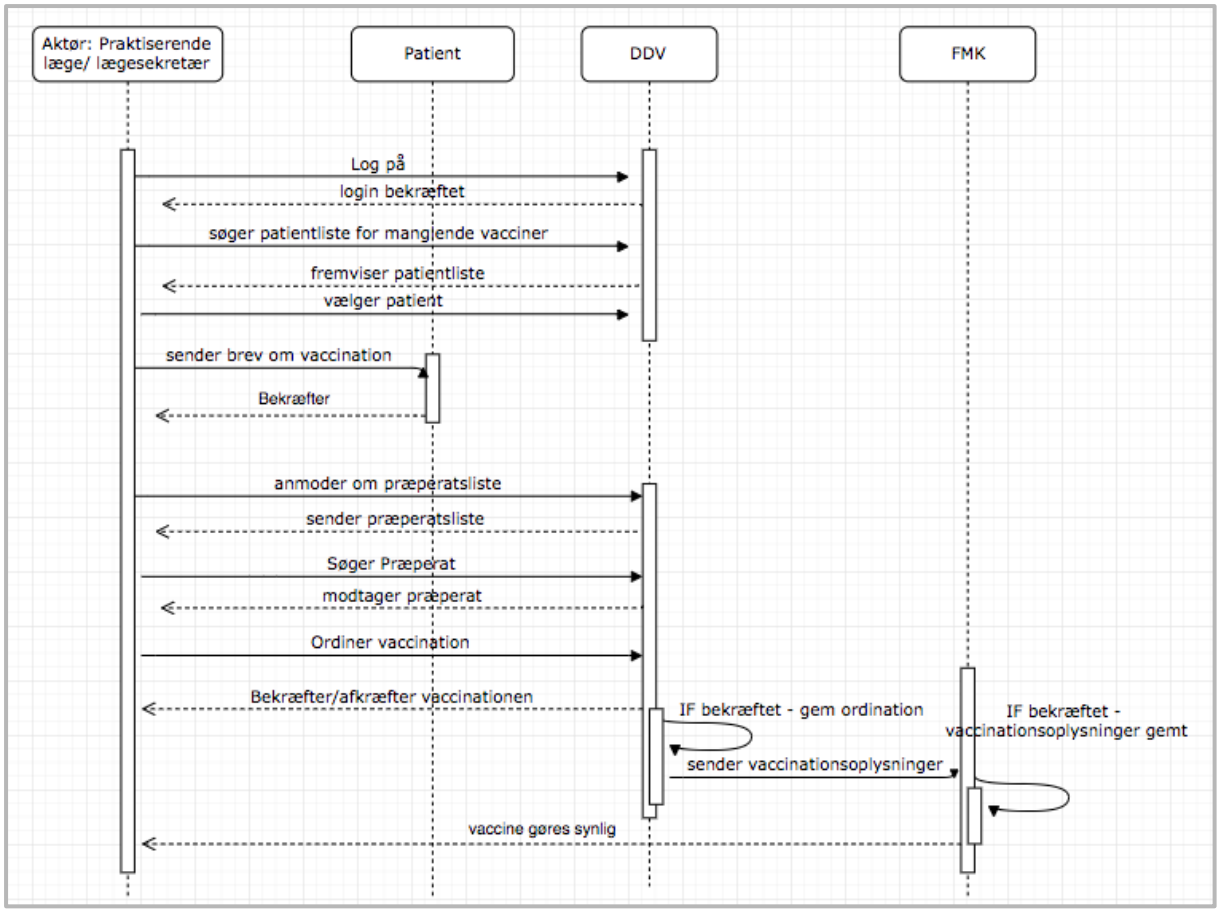


Bilag 11 Use case diagram



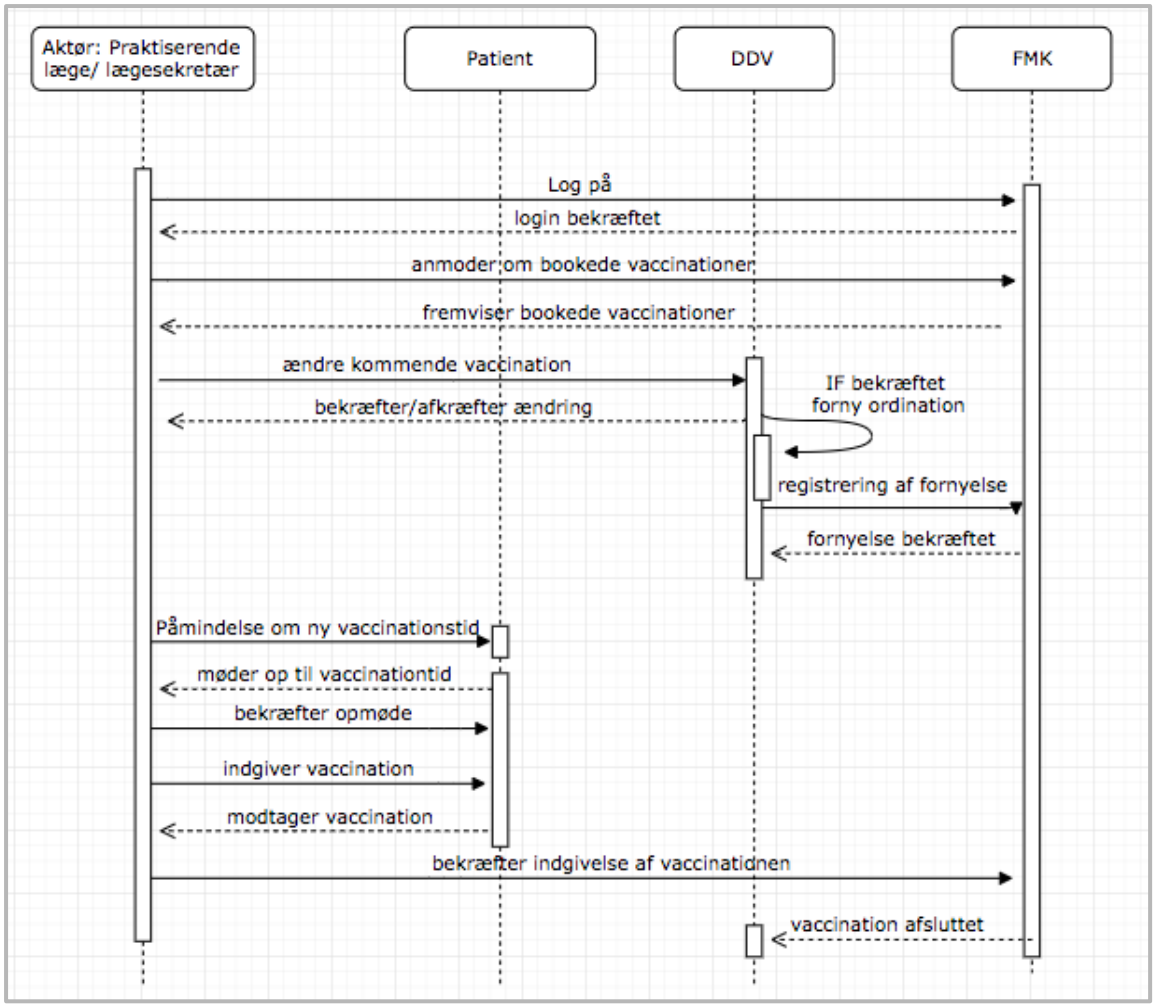
Bilag 12 Sekvensdiagrammer

Use case name: Registrering af vaccination



Bilag 12 Sekvensdiagrammer

Use case name: Indgivelse af vaccination



Use case name: Se vaccinationsdata