* Forside
* Indholdsfortegnelse
* Indledning
  + Introduktion
  + Problemformulering
    - Krav til opgav
      * Systems
    - Kvalifikation
    - Proces håndtering
  + Afgrænsning
    - Planlægning og realisation
    - Tid og erfaring
    - Reality vs opgaveformulering
* Metodevalg : i gruppe diskuterer om hvorfor vi vælger den. : why : intro
  + Processen
    - UP vs Vandfald
      * Fordel og ulemper
      * Vurdering
      * Hvordan bruger discipline
* Problemdomæne
  + : om vores grænsning og selv beskrivelse om projekt : midtTrafik flextur
  + Forundersøgelse
    - Risikoanalyse
    - dokumentanalyse
    - ooa vision use cases domain model : requirements
  + Kravspecifikation
* Procesbeskrivelse : how
  + Unified Process og MUST

Som vi valgte UP(unified process) for vore proces til den projekt, startede der forskellige faser og iterations planlægning på den 2 maj 2016. I starten valgte vi UP ved MUST (Metode til forUndersøgelse i Systemudvikling- og Teori herom), fordi de elementer, der indgår i MUST-metoden understøtter UP, og it-forundersøgelses teknikker og beskrivelsesværktøjer fra MUST angiver os konkrete ideer til proces og tidsplanlægning. Ifølge overblik over MUST-metoden (Bødker, Kensing, & Simonse, 2008, s. 30) kan man se, at MUST-metoden præsenterer et perspektiv på forundersøgelser, hvor man ser bredt på it-anvendelser. Derfor valgte vi MUST teknikker og faser ved UP for projekts planlægning. Selv om vi prøvede at inkludere MUST så meget som muligt, var der alligevel realistisk græns til vorte projekt. Projektet havde ikke en deltagelse af MidtTrafik, og derfor var det svært for os at inkludere MUST princip ’reel brugerdeltagelse’. Da vi lavede nogle teknikker, der involvere burgerdeltagelse, kunne vi ikke vælge ’virskomhedsbesøg’ og vi lavede ’interview’ med vores lærere og andre som kunderolle. Men der var alligevel muligheder fra MUST, som angiver os en bedre forståelse om produkt. For eksempel var ’dokumentanalyse’ en effektiv teknik for at analysere projekt. For princip ’en samlet vision’ og ’reel brugerdeltagelse’ brugte vi forskellige dokumenter f.eks. 1. årsprøven 2016 dokumenter (Iversen & Jensen, 2016) og Flexturs hjemmeside (MidtTrafik : Flextur, 2016). De dokumenter angiver os ideer om vores krav og mål. Ved dokumentanalyses kunne vi også planlagte forskellige algoritmer og designe mønstre workshop. Denne plan gav en stor chance for at lave mere realistisk og praktisk produktstyring og veldefineret interessentanalyse resultat. I inceptionsfase var der flere teknikker, som var brugte til processen, og vi vil diskutere dem i ”inception” afsnit.

* + - * proces:Tidsplanlægning

Lige efter den første møde på den 2. maj 2016, hvor projektgruppen accepterede opgaven, startede vores første tidsplanlægning. For projektstyrelse er der mange forskellige værktøj muligheder f.eks. Microsoft Project (kaldes kort for Project herefter), referencelinjeplan fra MUST. Til hvert værktøj er der fordele og ulemper til vorte projekt som 2-mandsprojekt. Ved UP skulle der mange iterationer hvor planlægning og ændringer involveres. Derfor startede vi en prototype af de meste brugte værktøj i vore klasse dvs. Project og referencelinjeplan. Project er en stærk værktøj, hvor man finder styrelse af tidsplan. Det angiver projektgruppen en overblik over hvornår aktivitet skal starter og hvor mange timer skal bruges. Men da vi prøvede prototype til inceptionsfase, var der mange redigering og tidsplanlægning til hver aktivitet, hvor vi ikke kunne bestemme præcis tidsrum. Og denne ændringsproces tog mere tid end aktuel inceptionsfase. På den anden side gav referencelinjeplan en overblik over aktiviteter, der bruges uden tidsmæssig stor diskussion, fordi de aktiviteter lagt som boble kunne flyttes og redigeres hurtigt. Selv om referencelinjeplan ikke kunne give os en præcis tidsplan til hvert aktivitet, valgt vi det for vores planlægnings værktøj for mere hurtigt og effektivt processtyring. I hver iteration kunne vi lave næste iterationsplan og hvert aktivitets boble var lagt tættere på hindanden på grund af deres type og proceslighed. Efter projektets iteration2 i elaboration kunne vi desværre se, at Project kunne være nyttig at vise hvert aktivitets starttid eller sluttid for konkret retningslinje. Derfor vil vi gerne prøve Project eller referencelinjeplan med mere specifik tidsplan til hvert aktivitet i fremtiden.

* + - * Mockups og prototype ved brugertest

Fra MUST kan man finde en teknik ’Eksperimeter med prototyper’ i d.4 fase dvs. fornyelsesfasen for at skabe erfaringer og konstruktivt anvende disse erfaringer i der videre arbejde (Bødker, Kensing, & Simonse, 2008, s. 321). UP har navnet prototype i implementations disciplin. UP inceptionfase bruger en eller flere protype og eksekverbare arkitektoniske prototyper i elaboration fase (Kruchten, 2003). Fra de to bøg kunne vi se, at forskellige typer af prototype og mock-ups ville være nyttige og nødvenlige. Prototype af ekstern system som flextur\_sats.jar eller okttp-2.5.0.jar, der angiver os en kilometer udregning fra google map lavede som prototype for at forstå bedre om arkitekturs og teknologiske problemer. Og tråd, hvor det kunne være nyttig til systems pris udregning, var også eksperimenteret som prototype. Derfor brugte ”eksperimental prototyping” på næsten alle faser, da usikkerhed og problemer opstod i vore produkt for at finde løsninger. Mock-ups kunne også vise os nogle koncepter til vore produkt og starte analysere funktionelle krav og behøv. Vi kunne også bruge horisontal mock-ups som horisontale prototyper til brugertest. Alle meget simple billeder af brugergrænsefladen præsenteredes til bruger i brugertest. Især vore målgruppe, der ikke er i besiddelse af smartphones (Iversen & Jensen, 2016), valgte nogle brugere inkluderet en især i over 70 årig. I testen brugtes mock-ups med scene-builder application template, hvor bruger kunne fulgte systemets brugergrænseflade. Selv om ved horisontal prototype kunne de ikke se, hvordan systemet skulle virker, gav de os en chance for at få adgang til brugernes konkrete erfaringer som MUST forklarer i den ’tænke-højt forsøg’ teknik (Bødker, Kensing, & Simonse, 2008, s. 235) Ulempen med den application-form af mock-ups forhindrede os at lave stor ændringer efter brugertest, selv om der var for mange knapper og tekstfelter, der ikke var så brugervenligt. Derfor brugte vi mere tid at læse testdokument fra brugertest og diskutere i gruppen for at lave ændring for brugervenlighed.

Der var mange vurderinger og analyser til vores planlægning, produkts behov og opgaveformål på grund af teknikker og aktiviteter fra MUST og UP. Vi brugte forskellige teknikker for de discipliner, imens nogle var fravalgt eller negligeret, fordi de ikke passede til vores gruppe og opgaves tilstand. For eksempel brugte vi review som walkthrough. Walkthrough kunne bruges til alle forskellige artefakter til produkt med tjekliste (Myers, 2004). I inception og elaboration (iteration 2) havde vi review med den andre gruppe på vores vision dokument, interessent analyse og use cases (både diagram og use case 1 og 2). Selv om den teknik var en stor fordel ved os at få nyttige tilbagemeldinger, var der ikke så mange, der kunne gøre review sammen i resten af processen. Derfor kunne vi ikke sætte den i de andre faser. Og der er nogle andre aktiviteter, der involvere et møde med MidtTrafik som vi navnede før, eller project management workflow til hver iteration. Det var også svært at følge ”buisness modeling artifact set” fra UP (Kruchten, 2003) på grund af vores gruppestørrelse og tilstand. Vi negligerede også nogle artefakter f.eks. ”stakeholder requests”, fordi der ikke faktiske nogle realistiske roller, der kunne deltage.

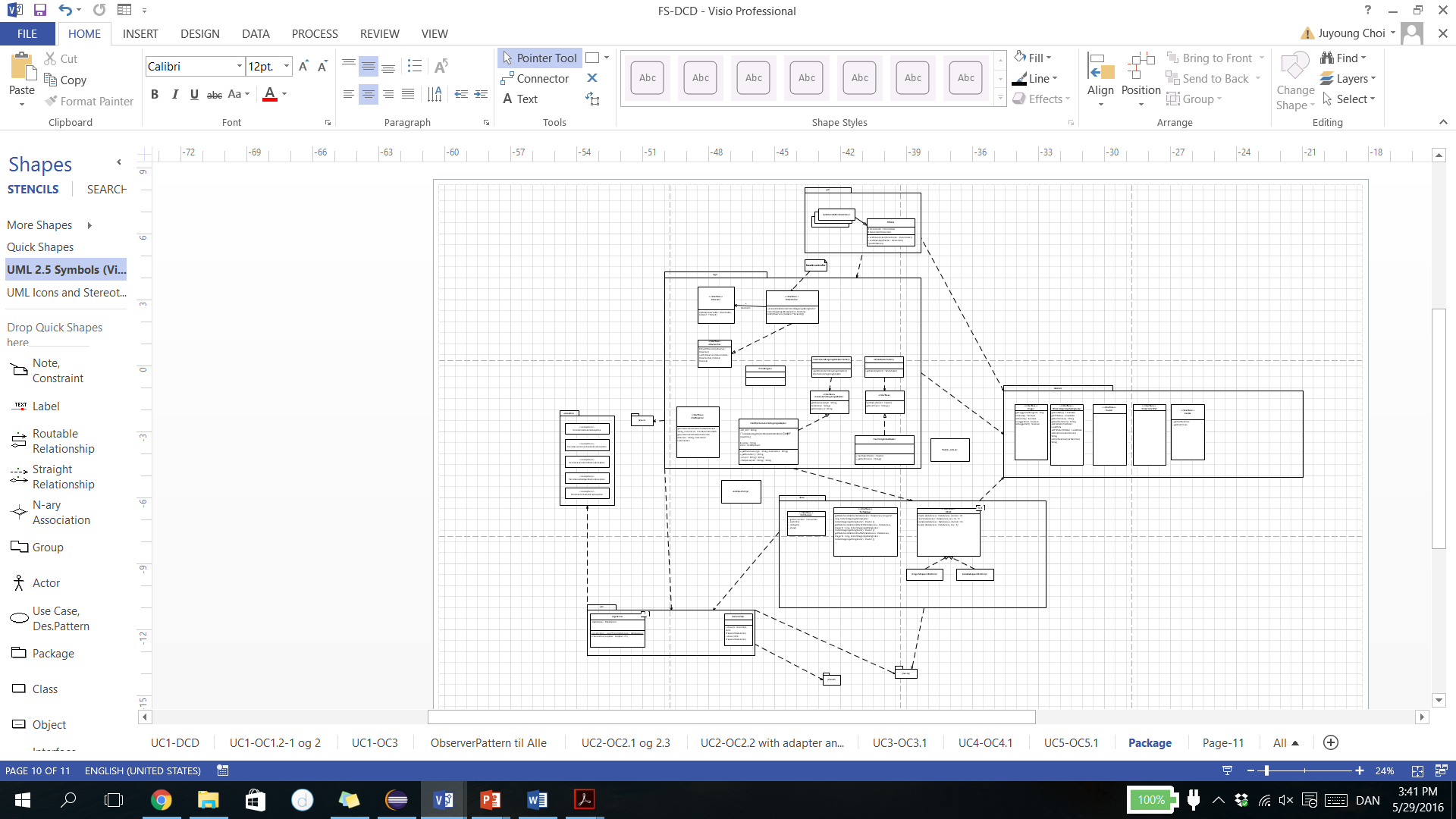
* + - Inception
      * Dokument Analysis
        + Proces
        + Vurdering
      * Brainstorm
        + Proces
        + Vurdering
      * Dataordbog (glossary)
        + Proces
        + Vurdering
      * vision dokument
        + Proces
        + Vurdering
      * Suppliemantary spcecification
        + Proces
        + Vurdering
    - Elaboration
      * Elaboration startede på den 10 maj 2016. Da den første use cases (UC1) systemsekvensdiagrammer blev færdig, kunne vi se om, hvor mange operationskontrakter og hvor lang tid skulle bruges. På grund af vores vurderinger sat vi 2 iterationer i elaborationsfase. Elaborations milepæl er navnet som ”Lifecycle Archtecture” (Kruchten, 2003). Det betyder, at når elaborations fase er færdig, vil der være forskellige tistande. For eksempel beskriver Anders, at der er uddybende kravspecifikation og implementation af centrale use cases, som er med højeste risiko og stor indflydelse på arkitekturen. Og derfor implementeres de vigtigste scenarier (Pederson, 2016). I vore elaborationsfase var alle use cases identificeret. Men der var alligevel muligt for udvidelse. Selvfølgelig kunne det være perfekt, da vi planlagte og lavede use case diagram og beskrivelse 100 percent korrekt fra staten. Men processen blev mere fleksibel og justerbar på grund af UP. Derfor kunne vi opnå både ændringer og vurdering af vores mistanker. I starten var der kun et eksternt system for prisudregning. Men Jonas Mørch fandt en løsning til afstand, der bruges til prisudregningen med sats. På den måde kunne vores kunder ikke har brug for at indtaste deres afstand. Okhttp-2.5.0.jar bruges til at hente afstandsoplysning direkte fra Google map. Men i inceptionfase var der ikke det som understøttet systemet. Denne ændring var nemmere at implementeres for os på grund af UP. Vore første use case, UC1, var ”Se historik” og den var klart til implementation i iteration2 og fuldt implementeret som milepæl beskriver i referencelinjeplan. ”Se historik” har 2 forskellige aktør, kunde og bestillingsmodtagelse. Da vi planlagte og før påbegyndte at beskrive operationskontrakt, så den ud som sammen funktion hvor hver aktør beder systemet om at vise historik. Fordi da vi startede at lave datamodel, var der kun ”flextur” som tabel, hvor historik oplysninger findes. Men da vi startede vores operationskontrakter, kunne vi se, den kunne være to forskellige use caser, fordi en er hver enkel historik til kunde, og en er historik som oversigt i en bestemte tidsinterval. Derfor tog vi en dag mere til iteration2 og skiftede referencelinje 2-1 til den 18 maj. Vi vurderede vores muligheder, hvor vi skulle ændre UC1 til to selvstændige use caser eller blive ved UC1 som vore første plan. Vi kunne se, at UC1 som en use case, kunne vi angive vores kunder, at begge to slags historiks søgnings muligheder. Det betyder, at kunde også har en nem og hurtig implementation af ”se historik” som en oversigt over deres total pris og ture. Lige nu var det ikke en funktionelle krav, men vorte system kunne være mere fleksiv at fange dem senere, fordi vi lavede den som enkel use case. Vi kunne desværre se, at den måske var en stor use case til systemet, og mangel of erfaring kunne være en årsag til det. Og vi vil diskutere om det videre i afsnit ”SQL og Database” i produktbeskrivelse.
      * Selv om Anders påpeger, at arkitekturen er stabil og velbeskrevet i elaborationsfase (Pederson, 2016), var der alligevel nogle problemer til systemet. På grund af vores mangel af erfaring og videnskab til design mønstre og tråd, kunne vi ikke indeholde god arkitektonisk grund i elaborationsfase. Der var to muligheder for os. En var, at vi implementerer UC1, UC2 og UC3 som vi planlagte og bare fortsætter uden design møsntre indtil vi finder nogle løsninger. Ellers kunne vi stoppe proces indtil vi kunne have god arkitektur. Problemet var, vi havde en dato til aflevering, og vi kunne ikke bare vente. Derfor sat vi mange forskellige workshop i den sidste periode af elaborationsfase og starte af contstructionfase, hvor vi kunne vurdere om, hvilke design mønstre kunne være velvalgte muligheder og hvordan de bruges i systemet, imens vi fortsat processen. Vi inviterede nogle andre datamatikere. Det betyder ikke, at de kunne give os løsning eller skrive nogle kode for os. Vi afprøvede vores teori og forståelse, og vurdere hvordan vi kunne håndtere det med dem, der havde bedre forståelse af systemet. Vi havde også tid til undersøgelse for at understøtte vores vurderinger og teori. Vi vil beskrive den proces i afsnit ” Produktbeskrivelse” senere. Desværre betyder det, at nogle aktiviteter i elaborationsfase ikke var helt færdigt som vi ønskered. Og der var en eller to dage, hvor vi kom tilbage til de use cases og fixede problemer senere i constructionsfase, selv om vi havde eksekverbare use cases. Men vi er også glad, at vi kunne finde løsninger og fik bedre strukturalt systemet.
    - Contruction
      * Contructionfase er for at implementere det resterende system. Vi havde 3 use cases, der var implementeret og de restende use cases var UC4, 5, 6, 7 og 8 i contructionfase. For os som førsteårsstuderende, var det meget stresset at se der over 60% ikke var færdigt. Hvis vi fulgte vandfaldsmodel, kunne der ikke være så mange tilbage. På den tidspunkt kunne der være alle OOA færdiggjort, og vi kunne have domænemodel eller designe af klassediagram (DCD) klart til lave interfacer. Ved UP kunne vi have en domain klasse, hvor alligevel mangler nogle instanser at bruge, fordi de restende use cases var manglende i DCD, da vi startede constructionsfase. Men det betyder også, at hvis vi finde bedre løsninger til oplysnings håndtering til use cases, var det nemmere at inkludere den uden påvirkning til systemet. Den proces gik faktisk hurtigere end vi forventede i starten. Selv om der var 6 use cases til implementation, var der ikke så højt risiko og funktionen var lignende til de første 3 use cases. For eksempel UC4 ”Registrer kørsel”s struktur var næsten samme som UC2 ”Bestil flextur”. Vores kode og klasser til systemet var allerede klart at bruge, og vi havde også bedre forståelse af implementationen i constructionsfase i forhold til i elaborationsfase. Sekvensdiagrammer blev også meget nemt at lave, fordi der var mange samme sekvensdiagrammer, der genbrugtes som referencer.
    - Transition
      * Transition er den sidste fase i UP. Den er til levering af software til brugerende med uddannelse af burger og fejlrettelser for vedligeholdelse af systemet (Kruchten, 2003). Derfor brugte vi for at tjekke vores produkt og rapport for en dag for alle aktiviteter i transitionsfase. Selv om det er lidt for kort for transition, sat vi kun en dag for at længere elaborations og constructionsfase. Hvis det var muligt, kunne det være nyttigt, at have 2-3 dage til transition.
* Produktbeskrivelse
  + Arkitektur

En af vores læringsmål understreger, at studerende kan anvende programmeringssproget til realisering af design mønstre. Og krav til produkt beskriver, at den anvendte arkitektur bør gate hensyn til, at systemet laves her og nu som enkelbrugerløsning, men ønskes på sigt flyttet til web-platform. (Iversen & Jensen, 2016) Derfor valgte og fravalgte nogle af designmønstre, som vi har lært i den første år, og vi vil introducere dem herefter.

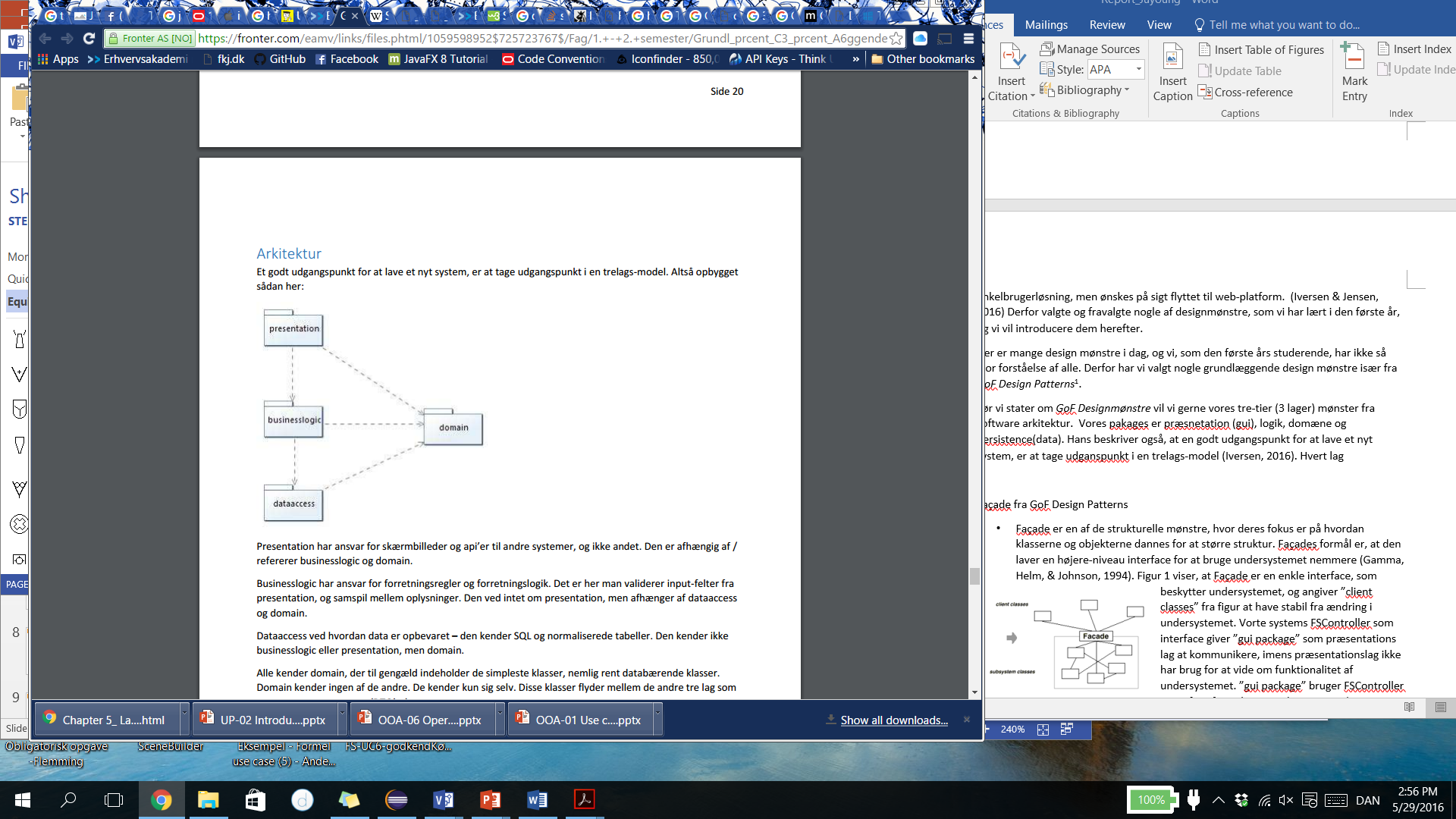
Der er mange design mønstre i dag, og vi, som den første års studerende, har ikke så stor forståelse af alle. Derfor har vi valgt nogle grundlæggende design mønstre især fra *GoF Design Patterns*[[1]](#footnote-1).

1. Trelags-Model og MVC model

Før vi stater om *GoF Designmønstre* vil vi gerne vores tre-tier (3 lager) mønster fra software arkitektur. Vores pakages er præsnetation (gui), logik, domæne og persistence(data). Hans beskriver også, at en godt udgangspunkt for at lave et nyt system, er at tage udganspunkt i en trelags-model (Iversen, 2016). Hvert lag har sit ansvar. Alle kender domæne package, men domæne kender ikke de andre lager.



presentation



dataaccess

domain

businesslogic

Figur 2Trelags-model fra FlexturSystem matchende til Fitur 1

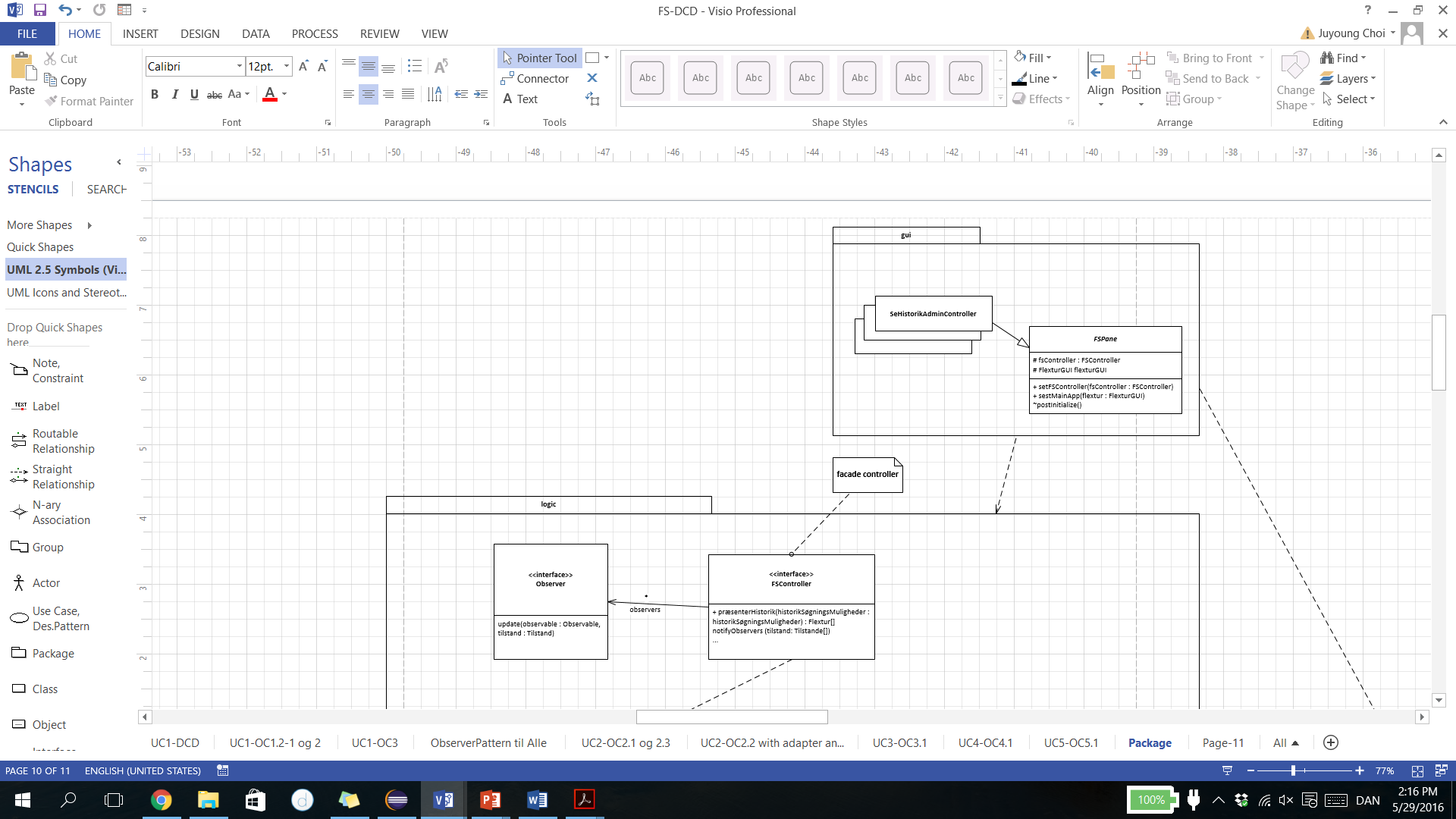
Figur Trelags-model.

Figur 1 viser, at præsentation sender besked til logik, og den er afhængig af logik og domæne. Logik er for forretningsregler og forretningslogik. Det lag validerer input-felter fra præsentation, men ved ikke om højere lag (dvs. præsentationslag). Dataaceess kender kun domæne, og har ansvar til at kommunikere med data og SQL.

Trelags-model angiver også en mulighed af ”Model-View Separation Principle” Domæneobjekter[[2]](#footnote-2) (dvs. model) i logiklag har ansvar til at fylde oplysninger og ”lower representational gap” mellem virkeligheden og software design, mens præsentation(UI) som view bruges til brugergrænseflade og for at få brugers input og oplyse resultatet. Observer mønster bruges til denne ”Model-View Separation” disciplin, hvor domæneobjekt sende besked til UI objekt (Larman, 2004), og controller kan være en mulighed for obsevable (eller subject) og vi vil diskutere den videre i afsnit ”Observer mønster”

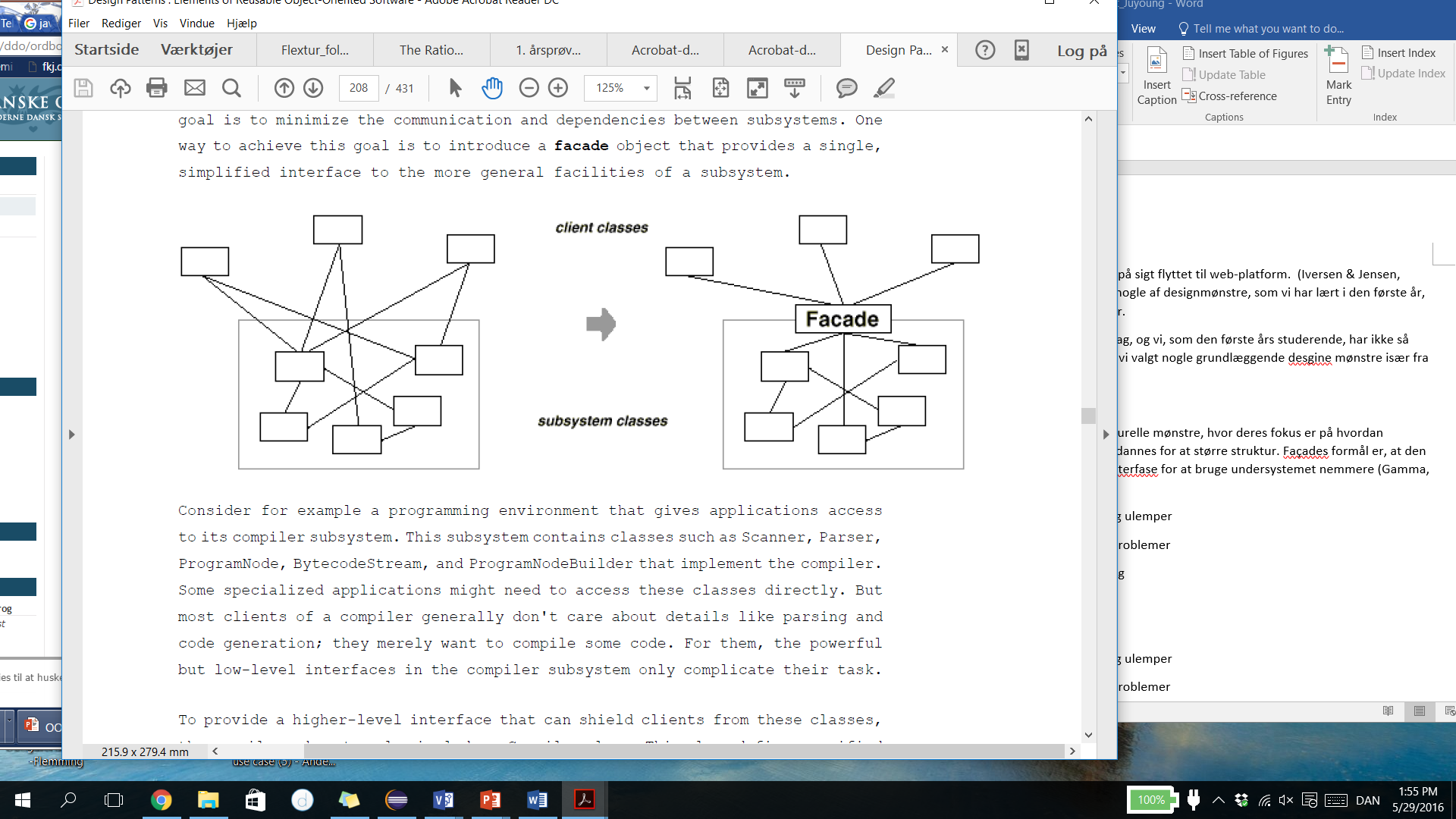
1. Façade fra GoF Design Patterns

Façade er en af de strukturelle mønstre, hvor deres fokus er på hvordan klasserne og objekterne dannes for at større struktur. Façades formål er, at den laver en højere-niveau interface for at bruge undersystemet nemmere (Gamma, Helm, & Johnson, 1994).



Figur 3 Façade fra GoF Design Patterns

Figur 4 Façade : FSController interface



Figur 3 viser, at Façade er en enkle

interface, som beskytter undersystemet, og angiver ”client classes” fra figur at have stabil fra ændring i undersystemet. Vorte systems FSController fra figur 4 viser, at FSController som interface giver ”gui package” som præsentations lag at kommunikere, imens præsentationslag ikke har brug for at vide om funktionalitet af undersystemet. ”gui package” bruger FSController interface for at kommunikere og sende metodekald. FSController vil bare sende videre til undersystemet for eksempel til ”data package” som persistencelag hvor TurMapper eller KundeMapper handler kommunikation med data. Imidlertid ved vores gui controllere ikke om hvordan data og list af Flextur hentes. De bruger kun FSController, der er en façade controller. Da vi ændrede nogle linjere af kode i logik eller data package (persistence lag), holdte gui alligevel stabil på grund af façade.

1. Singleton
   * + - 1. Fordel og ulemper
         2. Tekniskproblemer
         3. Vurdering
       1. Observer pattern
          1. Fordel og ulemper
          2. Tekniskproblemer
          3. Vurdering
       2. Factory pattern
          1. Fordel og ulemper
          2. Tekniskproblemer
          3. Vurdering
       3. Adapter pattern
          1. Fordel og ulemper
          2. Tekniskproblemer
          3. Vurdering
       4. thread
          1. Proces
          2. Vurdering
          3. Evaluatering
       5. problem håndtering
          1. Exception
   1. Produkt
      1. OOA
         1. Aktivititetsdiagram og Usecase
            1. Proces
            2. Vurdering
            3. Evaluatering
         2. Testcase
            1. Proces
            2. Vurdering
            3. Evaluatering
         3. Datamodel
            1. Proces
            2. Vurdering
            3. Evaluatering
         4. SSD
            1. Proces
            2. Vurdering
            3. Evaluatering
         5. DM
            1. Proces
            2. Vurdering
            3. Evaluatering
         6. OC
            1. Proces
            2. Vurdering
            3. Evaluatering
      2. OOD
         1. SD
            1. Proces
            2. Vurdering
            3. Evaluatering
         2. DCD
            1. Proces
            2. Vurdering
            3. Evaluatering
      3. MUST og GRASP
         * 1. Proces
           2. Vurdering
           3. Evaluatering
      4. Implementation
         1. Coding with lambda for LogicTrans
            1. Beskrivelse
            2. Proces
            3. Vurdering
            4. Evaluatering
         2. Data with CRUD
            1. Proces
            2. Vurdering
            3. Evaluatering
         3. SQL
            1. Proces
            2. Vurdering
            3. Evaluatering
         4. Især design af SQL var meget svært, da nogle af ”sql query” ikke kunne passe til domain og persistence del af systemet. For eksempel var vi startede at prøve at hente historik fra database, kunne vi se, at flextur som domain ikke kan indeholde
         5. External System with jar.
            1. Proces
            2. Vurdering
            3. Evaluatering
         6. JUnitTest
            1. Proces
            2. Vurdering
            3. Evaluatering
         7. CSV fil Exportering
            1. Løsninger : med javafx afhængigheder
            2. Vurdering
2. Konklusion
3. Litteraturliste
4. Bilag

1. *GoF Design Patterns* : *Design Patterns* er en bog, der normalt kaldes som Gang of Four (eller GoF) design mønstre. [↑](#footnote-ref-1)
2. Den domæneobjekt er fra [Larman04] er en software objekt som logisk repræsentation, mens domain fra *Figur 1* har ansvar som Information Expert fra *GRASP* og businesslogic indeholder controller f.eks. FSController. [↑](#footnote-ref-2)