Prototype

Linear Approach
Works well for stable and explicitly stated problems

- Inability to respond efficiently to change
- When things are not stable. When you discover new things along the way and the previous understanding is no longer relevant.

Avoid Waterfall method with prototypes

- There can be overlasp

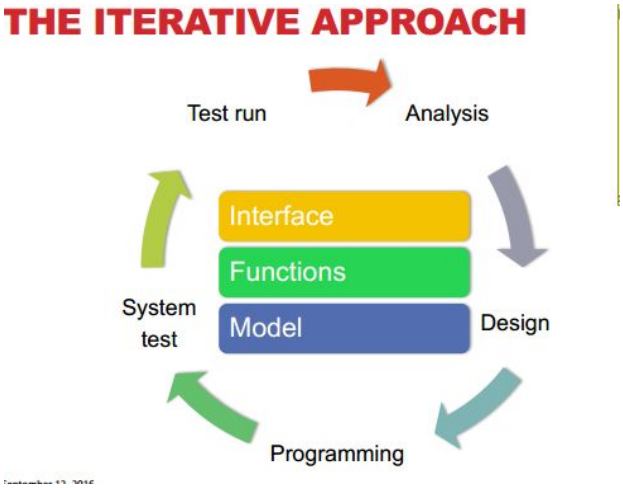
Users aren't always right when building an application, look usability

Real problems are rarely clear and precise

1. Try out things and see how it works out, trial and error.
2. Emphasizes uncertaintees.
3. Seeking a satisfactory solution/version.
4. Version development is in close communicaton with the specified users.
5. Hard to know when you're done since everything can be improved.
6. Can sometimes happen that the problem is **different** than what users actually want.

Evolution recognizes and emphasizes the uncertainties

- The problem is interpreted and restated

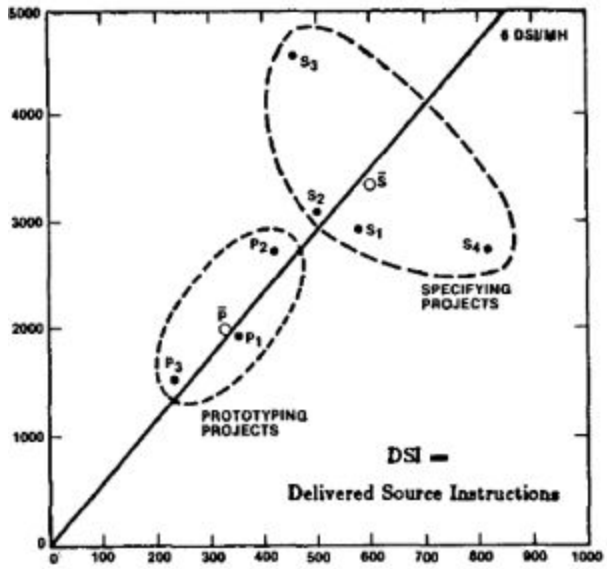


Devs can do **system test** while users will do **test runs**

Boehms experiment

7 groups solve the same set assignment:

1. Specification-oriented == Construction, 4 people
2. Prototypes == evolution, 3 people
3. We can see that teams that used Prototypes delivered a lot faster and a lot simpler solutions.



Construction = more time spent
Prototyping = less time spent

- Not a lot of Design Logic applied
- Less functionality and less robust than Construction method.
- Approach should be chosen on what the goal is.

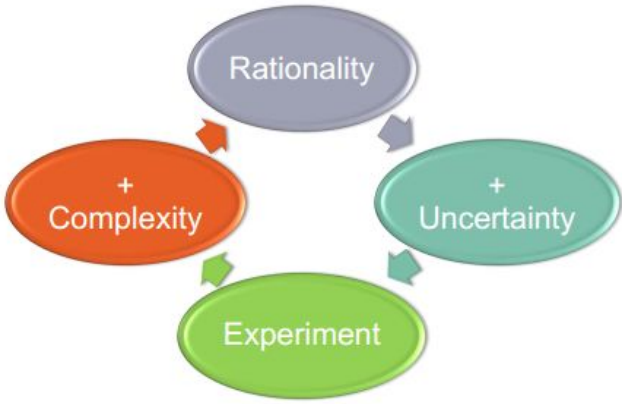
Mechanistic	Romantic
• Algorithmic (Church-Turing) • Rational manipulation of symbols (Descartes) • "Machines can think – when we have completed the programs" • Systems development is **construction**	• Competences and knowledge is richer than information and data • Information = data + interpretation • "Machines will never think" • Systems development is **evolution**

Uncertainty and complexity

1. Complexity described as:
Algorithmic and Rule based, look at algorithms structure.
2. We divide the problem and put all smaller solutions together for a final solution.
3. However not great at handling uncertainties

4. Uncertainty is usually user feedback on a problem that can be misconstrued or handled wrongly.
5. Users can also misunderstand the problem.

6. If both are not sufficient ways for a workflow you can mix and match until you find what works best.



1. Look at the common problems and evaluate.
2. We do however risk losing the important things when mixing and matching.

Concepts

Users:

• Work with or uses the application system (may not just be people looking at a screen or using a mouse).

- Users can be customers, employees and so on.
- When developing the prototype you exchange thoughts with users and we both learn something.

Client:

• Initiates, makes agreements and signs the contract with a software manufacture

Launching:

1. Presentation is developed quickly to give a quick future representation of what the future proto might look like. Usually not a lot of functionality.

2. Proper is only to display specific functions for consideration. A lot more realistic than the presentation.

3. Pilot is a prototype that is actually used for experimenting. Peak realism.

Actual prototyping:

Paper prototypes:

Communication with user and dev

Powerpoint prototype

- Having a PowerPoint presentation that can be moved through by a user. Gives impression of what the system might look like.

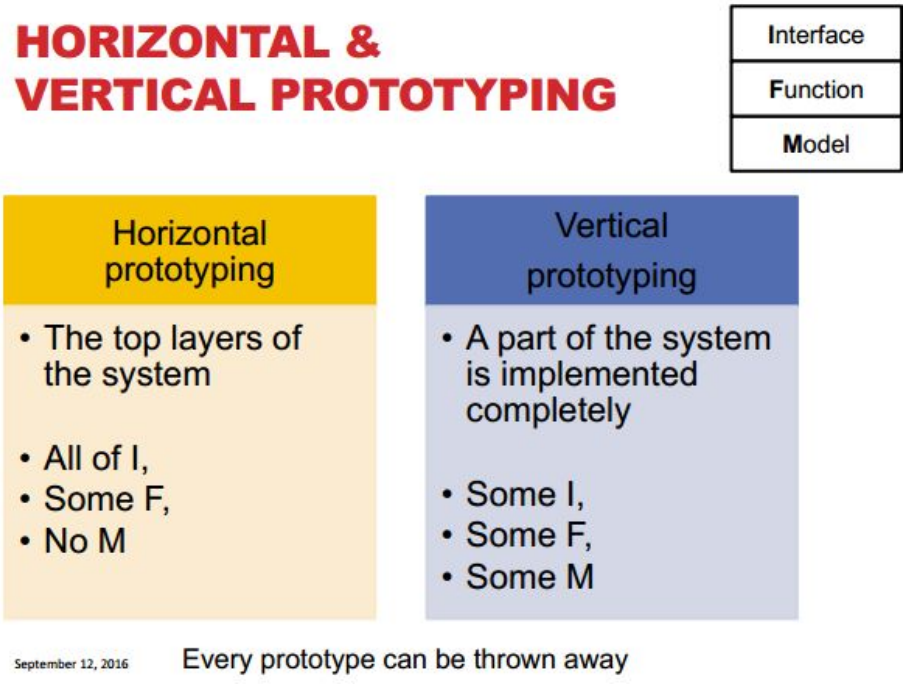
General:

1. Prototyping can give access to learning something when engaging in our analysis.
    - What is the problem? Explore the problem domain.

2. Experimental gives us the ability to refine and tweak our design. Can have different prototypes doing something different to compare.

3. Did we understand the problem correctly, is this the right direction?

4. Evolutionary is a process that is gradually evolving and can be refined a lot for acclimating.



1. Horizontals main focus is on the interfaces. Like the powerpoint examples.

- Might need to include some functionality.
- Not concerned with data models.

2. Vertical does not focus on one layer but on specific functionality.

- A part of the system is implemented.
- You can use this along the way to steer on the real product.
- What did you learn.

Principles:

When a prototype becomes something the Users want it becomes a part of the requirements. Prototypes are ONLY steps along the way.

Cooperative Interaction:

With this method you can analyze user input.

- You can see when a user expected something else from a functionality.
- Opens up for a constructive dialogue between user and dev.

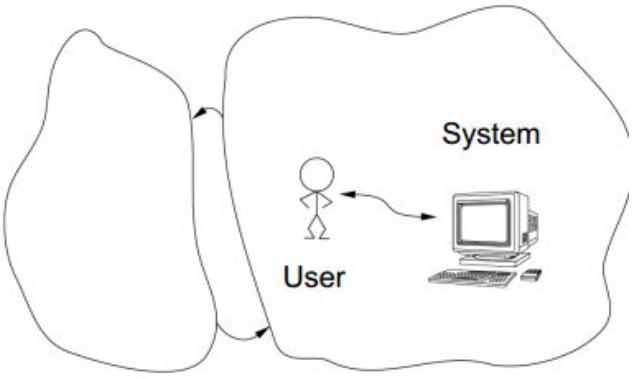
There is a risk that the user stops interacting with the interface and starts interacting with the wizard/you.

System Choice

THE SYSTEM CONTEXT

Problem domain:
That part of a context that is administrated, monitored, or controlled by a system.

Application domain:
The organization that administrates, monitors, or controls a problem domain.



A problematic situation can be interpreted in many ways.

System choice is about defining the system.

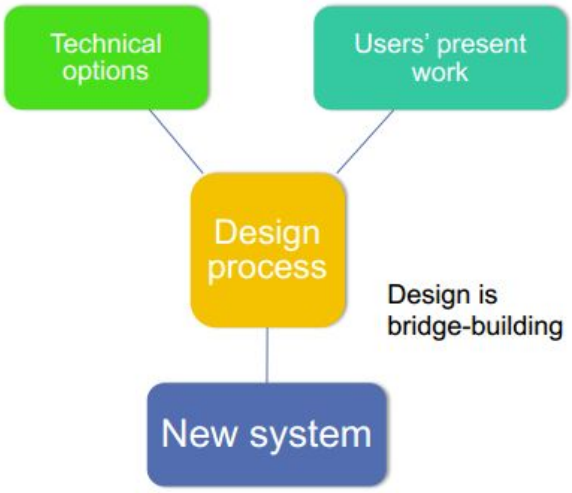
September 19, 2016

In collaboration with the users!

Abstract and Concrete experience of the user:

1. Example of concrete experience could be being a University student, knowledge about going to school is only gotten through experience and not knowledge gained from home.

2. You had abstract knowledge about this before coming to university. Knowledge about the problem domain is abstract at first, then concrete.



Techniques and Tools:

T & T	Areas of knowledge
Observations	1
Self-registration	1
Mock-ups	1, 6
Rich pictures	1, 2
Culture analysis	1, 2
OO design	5
Future workshop	2, 5
Card games	1, 6
Visits	3, 4
Study standard sw.	3, 4

	Users work	New system	Tech. options
Abstract knowledge	2	5	4
Concrete knowledge	1	6	3

1. Concrete experience - user's work
2. Relevant structures on user's work
3. Concrete experience - tech. options
4. Overview of technological options
5. Visions and design proposals
6. Concrete experience - the new system

T & T	Areas of knowledge
Interviewing users	1, 2
Video recording	1
Think-aloud exp.	1, 6
Conceptual modeling	2
OO analysis	2, 5
ER-diagrams	2, 5
Metaphorical design	2, 5
Prototyping	3, 5, 6
Literature study	4
Forum theater	6

	Users work	New system	Tech. options
Abstract knowledge	2	5	4
Concrete knowledge	1	6	3

1. Concrete experience - user's work
2. Relevant structures on user's work
3. Concrete experience - tech. options
4. Overview of technological options
5. Visions and design proposals
6. Concrete experience - the new system

	User's present work	New system	Technological options
Abstract knowledge	Relevant structures on user's present work (2)	Visions and design proposals (5)	Overview of technological options (4)
Concrete knowledge	Concrete experience with user's present work (1)	Concrete experience with the new system (6)	Concrete experience with technological options (3)

1. These areas of knowledge are segregated based on how that knowledge is gained.

2. They can overlap easily. For example with an observation and doing prototype work with the user.

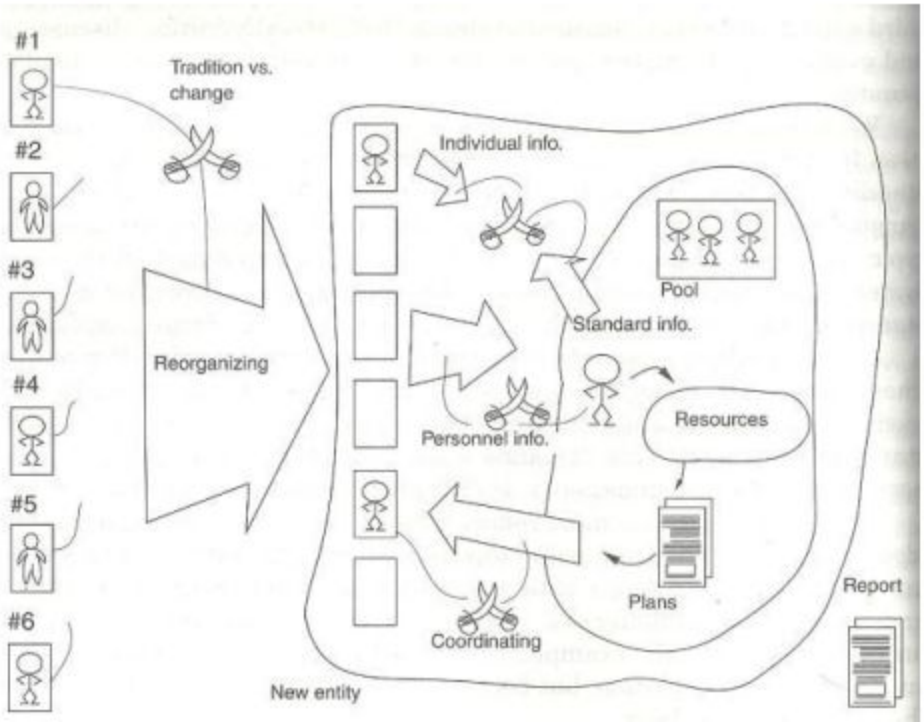
Rich Picture: an informal drawing that presents the illustrator's understanding of a situation.

- Focuses on change or stability
- Focuses on important aspects
- Gives a broad description
- A tool to help system developers organize their understanding
- Faciitates interaction between users and developers

1. It can focus on change and context of current or future situation. or stability.

2. Need to go out to the users and collab with rich pictures.

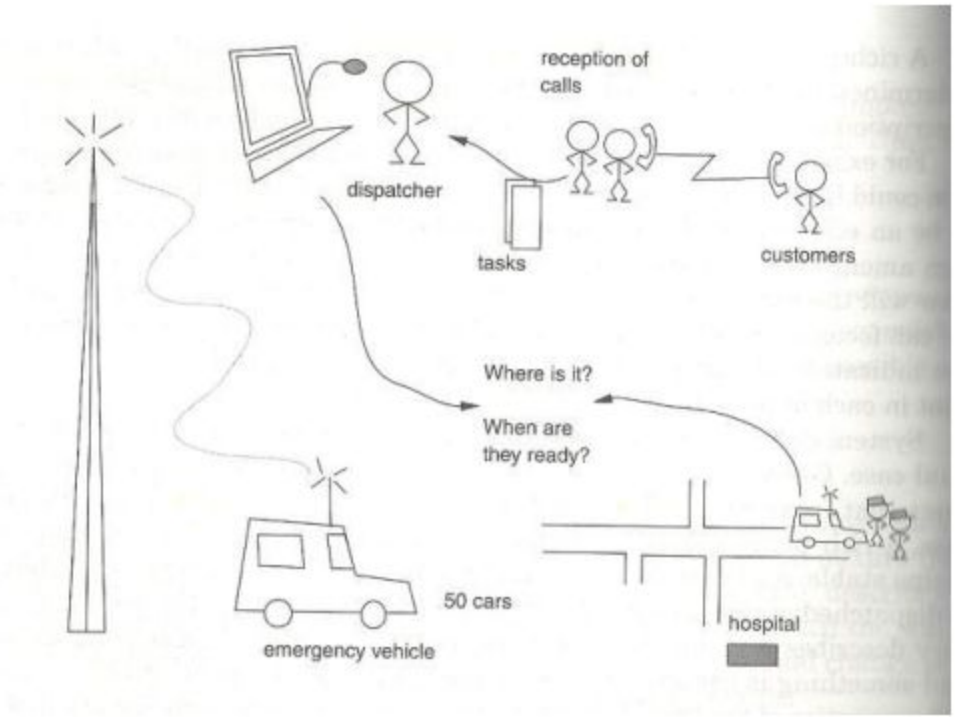
RICH PICTURE WITH FOCUS ON CHANGE



1. 6 independent wards. merge into one unit.

2. pic is trying to illustrate this change, no technical solution to the problem. Only the understanding of the changes.

RICH PICTURE WITH FOCUS ON STABILITY



1. How does the em vehicle dispatch working?
Very general overview of a situation as well.

DRAWING RICH PICTURES

- **Entities:**
  - People and places,
  - Roles and tasks tie people together
- **Processes (use arrows):**
  - Work and production,
  - Information processing (how people use information to interact),
  - Planning and control,
  - Development and organizational change
- **Structure (use lines or place within figures):**
  - Production and application,
  - Communication and agreements,
  - Ownership,
  - Membership,
  - Power relations

1. Focus on entities, things we can observe, places people etc.

2. Focus on who needs to interact in order to perform the diff work processes. Would use arrows in a picture.

3. Processes related to planning, development, interactions.

4. By grouping structures together we show how different entities communicate and work together. Who owns the process, membership as a part of the process and power relations, (who makes the decision) might need to include some and not all depending.

Cultivate new ideas:

1. After grasping the situation start working on ideas.

2. Important part is to challenge existing traditions. Do WE want to reproduce the current system or innovate a new system that has better traditions.

3. Experiment with prototypes to evaluate new ideas based on improvements you would make.

Create new ideas:

1. What has others done before to solve these problems?
2. Shelf-ware can be tailored to our needs. SAP is an example of this. A program made of modules that have tight integration.
3. Extremely costly to implement such a system.
4. Can use metaphors, things can be seen as other situations.
5. Physical locations can generate ideas of what a metaphor or system design could look like.
6. This is only to give ideas of how the system MIGHT look like not what it will be like.

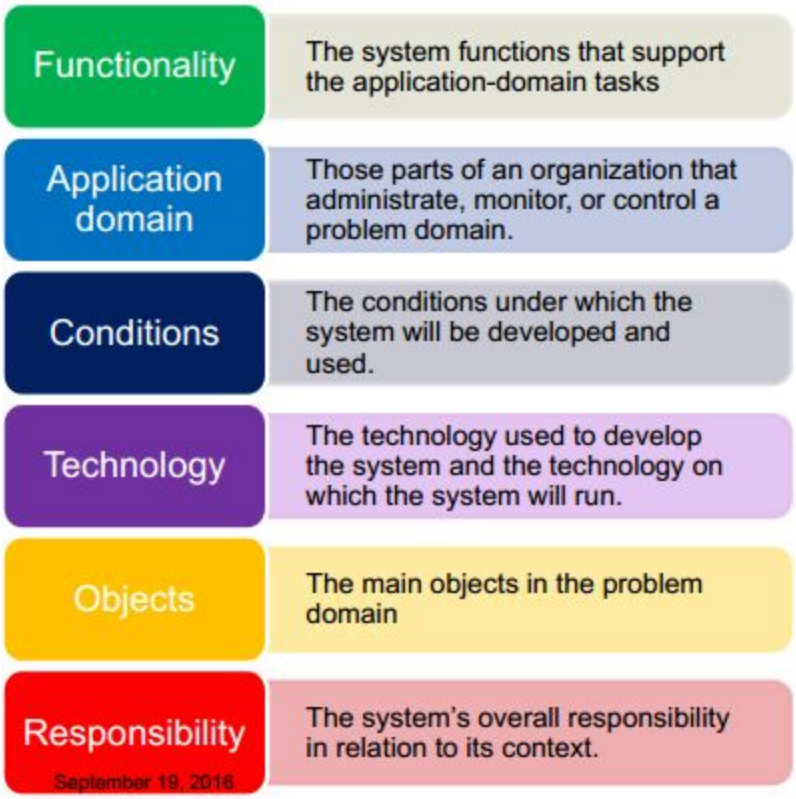
Systemdefinition

1. A brief description of what the new system should be. There can be multiple description but with different focus.

Fra rapport Systemdefinition: Web applikation

Et informations- og kommunikationssystem med administrative værktøjer til VUC&hf. Systemet gør det muligt at håndtere informationen omkring kurser, lektioner, afleveringer, fravær, lærer og kursister. Det skal være muligt for lærerne at formidle lektier og afleveringer til kursister gennem systemet. Systemet skal være kompatibelt med en bred række af computere, og brugerfladen skal dermed være fleksibel. Udviklingen skal foregå i tæt samarbejde med informanten.

BATOFF: Web applikation
Betingelser Udvikles i samarbejde med informant.
Anvendelsesområde Administrationen og lærere, planlægningsværktøjer.
Teknologier Systemet vil benytte sig af en server med en database. Brugerne benytter en bærbar, stationær eller smartphone.
Objekter Kursister, Lærer, Administratorer, Lokaler, Kurser, Lektioner, Materialer, Afleveringer.
Funktioner Administrativt værktøj der bruges af kursister, lærere og administrationen til at kommunikere og dele information.
Filosofi Informationssystem og administrativt system

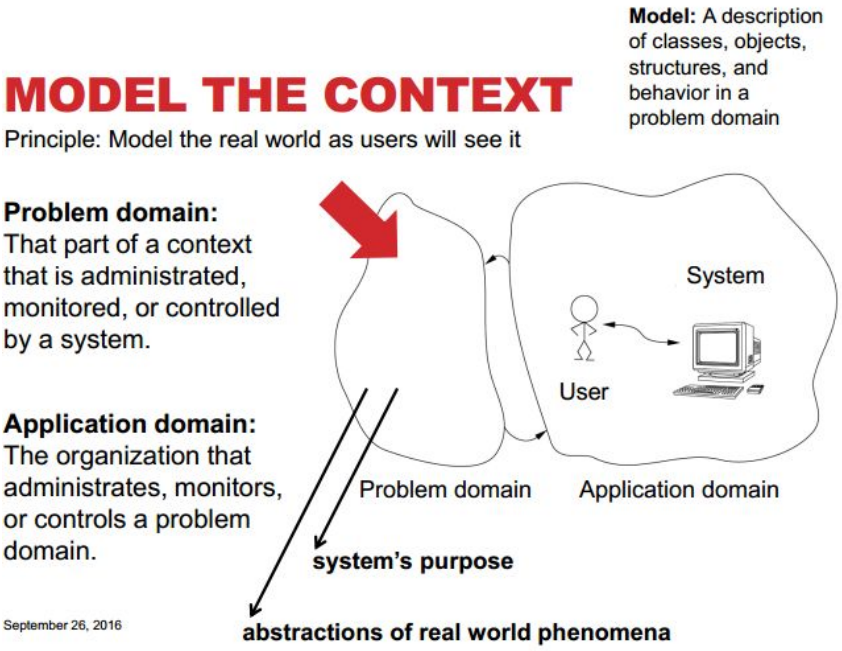


1. Alternate systems can all be defined with the FACTOR checklist.
2. Alternate system definitions can display different focuses on the same problem and application domain.
3. We create equally viable alternatives so the users have a real choice. If they have no choice WE made the choice for them = not good.

Choose a systemdefinition:

1. Our job is facilitating these systems.
2. We need to qualify the users to be able to choose between these alternatives.
3. It's important when this system is chosen that its seen on a whole scale. Not just one definition.
4. Only choose ONE definition for the design.
5. Avoid reproducing old ideas

Modeling Classes

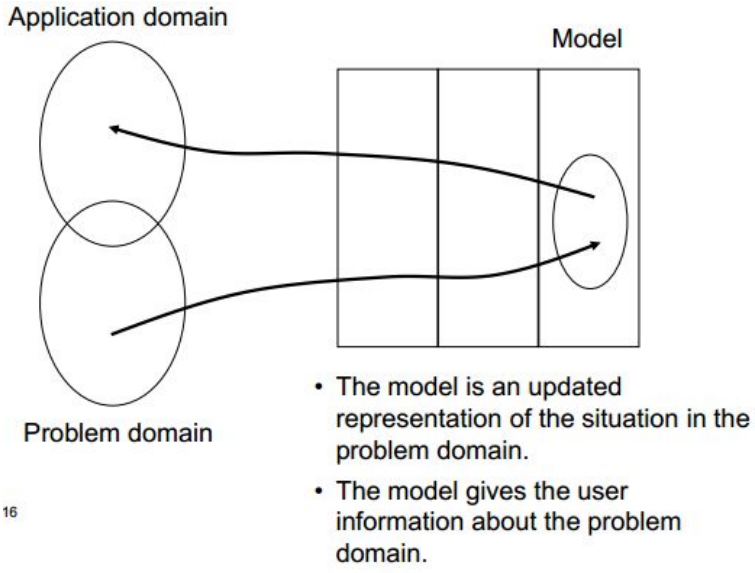


Modelling everything that has an impact on the use-situation

1. If something changes in the real world it should be reflected in our model.

2. We must keep an eye on changing information in the problem domain in order to adapt our model

3. The loop here represents this.



Analysis

ACTIVITIES IN PROBLEM-DOMAIN ANALYSIS

Activity	Content	Concepts
Classes	Which objects and events are part of the problem-domain?	Class, object, and event
Structure	How are classes and objects conceptually ties together?	Generalization, aggregation, association, and cluster
Behavior	Which dynamic properties do the objects have?	Event trace, behavioral pattern, and attribute

Principle: First get an overview, then supply details

1. Classes - Which objects and events are part of problem domain

2. Structure - How do they tie together

3. Behavior - How do they change

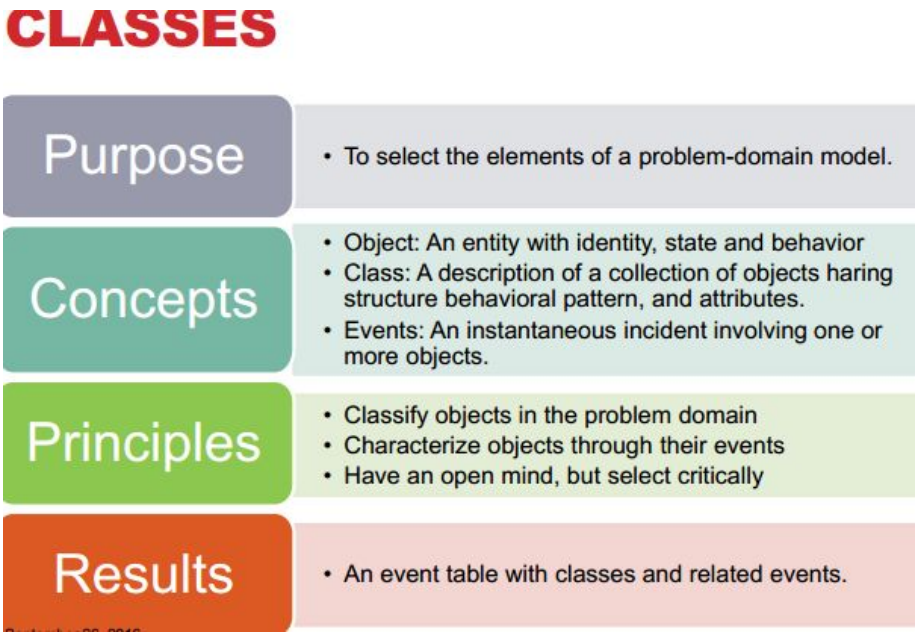
Choosing events and classes and objects:

1. We are not looking at activites that unfold over time, but events that happen instantaneously

1.1 The difference is the time frame

2. We characterize objects through events.

2.1 Be critical in choice of events.



Classify objects and events:

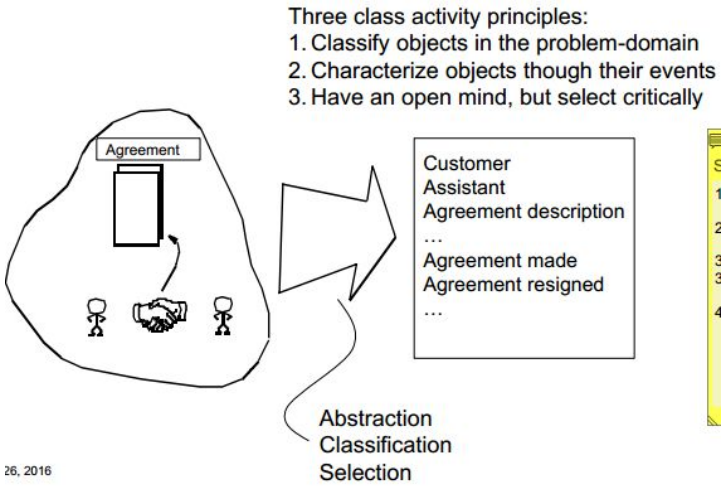
1. When we classify we make generalization.

2. We only identify through important characteristics.

3. Similarly with events, some events are expressions of the same thing.

3.1. By generalizing and making abstractions we make ourselves aware of this.

4. Model here shows how events and activities and objects coincide with each other.



OUTCOME OF 'CLASSES'

Example: hair salon

	Classes				
Events	Customer	Assistant	Apprentice	Appointment	Plan
reserved	X	X		X	X
cancelled	X	X		X	
treated	X			X	
employed		X	X		
resigned		X	X		
graduated			X		
agreed		X	X		X

1. The events in this example. These are relevant to the system.
2. The next step as evident in the cross-outs, is to tie the event types to classes.
3. This model is to model the most centric ideas behind the problem domain, only still focusing on the problem and no the solution which would have way more classes.

Eksempel fra rapport, klassediagram:

	Lærer	Kursist	Sekretær	Kursus	Lektion	Lokale	Nyhed	Besked	Aflevering	Afleveringsbeskrivelse.
begyndt	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
aflyst				✓	✓				✓	✓
ændret				✓	✓		✓		✓	✓
forladt	✓	✓	✓							
besked sendt	✓	✓	✓					✓		
besked modtaget	✓	✓	✓					✓		
tilmeldt	✓	✓		✓						
afmeldt	✓	✓		✓						
reserveret					✓	✓				
afleveret		✓							✓	✓
udløbet				✓					✓	✓
fravær givet	✓	✓			✓					✓
karakter givet	✓			✓					✓	

Table 5.1: Hændelses- og klasse diagram på programmet.

Finding classes:

- **Generate a candidate list**
  - Make a list of all potentially relevant classes
- **Consider many sources**
  - Your own perception of the problem domain
  - Existing descriptions (rich pictures, the system definition, etc.)
  - Collaborate with prospective users
- **The names for the candidate classes must be**
  - Simple and readable,
  - Originate in the problem domain, and
  - Describe a single instance

Can draw classes from system definition and BATOFF

Finding events:

- **Generate a candidate list**
  - Make a list of all potentially relevant events
- **Consider many sources**
  - Your own perception of the problem domain
  - Existing descriptions
- **Eliminate verbs related to the way users carry our their job**
  - These belong to the application domain
- **The names for event candidates**
  - Must be simple and readable
  - Originate in the problem domain, and
  - Indicate a single event.

1. After determining potential classes you start identifying events.

2. These only belong to the problem domain.

Critical selection:

1. When the list is created we make our selection which is entirly critical.

2. Is the class or event relevant to the problem domain? No, then exclude.

3. Identifying objects through a class is essential to its relevancy.

4. Normally you want to avoid redundancy, hence don't include classes or events that deal with the same deal.

5. Classes should internally be cohesive

6. Distinguish between events and activities (Activities are instant, events are gradual)

7. Can the event be identified? If not we might create a problem we don’t need.

Base this critical selection on your system definition.

Modeling Structure

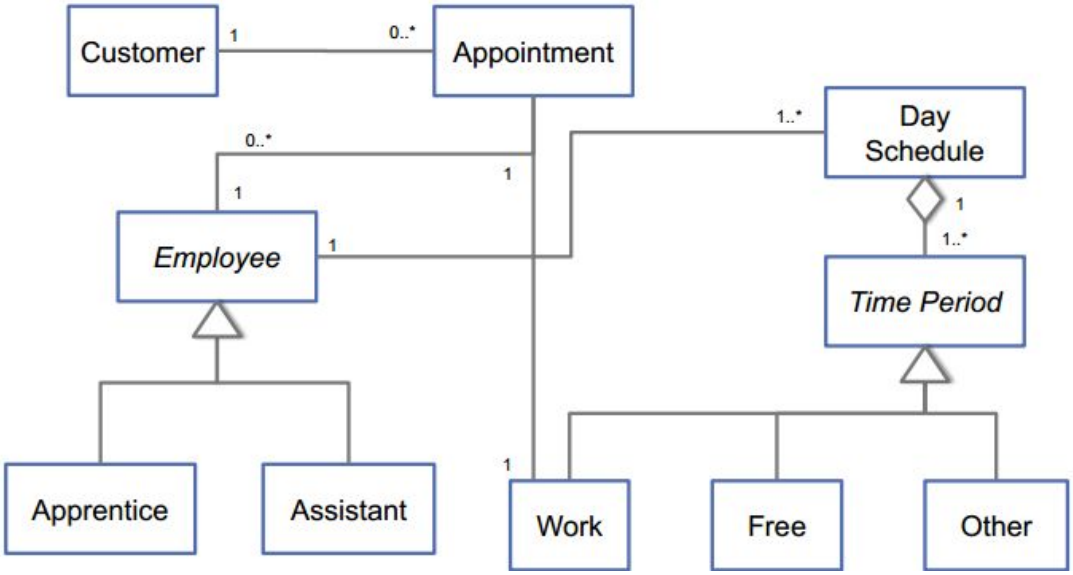
1. First look for candidates via relations between classes.
2. Generalizations and cluster relate to the static relations between objects.
3. Explore patterns
- General solution to generalized problems

4. Evaluate systematically
- Every structure must be used correctly

ELEVATOR – EVENT TABLE

- **Context:**
  - The system is used to decide where an elevator must stop based on the requests of users; e.g. "go to floor 3". Other systems control the movements of the elevator, its speed, direction and position between floors.

	Elevator	Floor	Choice	Call down	Call up
Left down	X	X		X	
Left up	X	X			X
Arrived	X	X	X		
Called down		X		X	
Called up		X			X
Floor chosen			X		



Eksempel fra rapport, klassediagram struktur

5.2 Struktur

For at beskrive relationerne mellem klasserne fundet i afsnit 5.1.1, er der lavet et klassediagram som kan ses på figur 5.1. Formålet med klassediagrammet er at give et overblik over problemområdet.

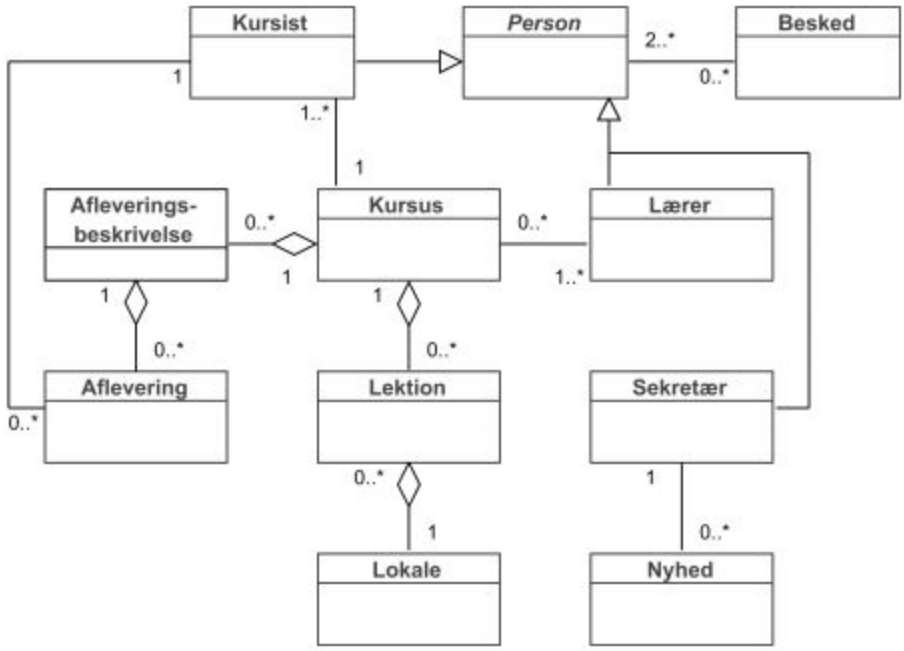


Figure 5.1: Klassediagram for problemområdet

Er en "is-a" eller Generalization

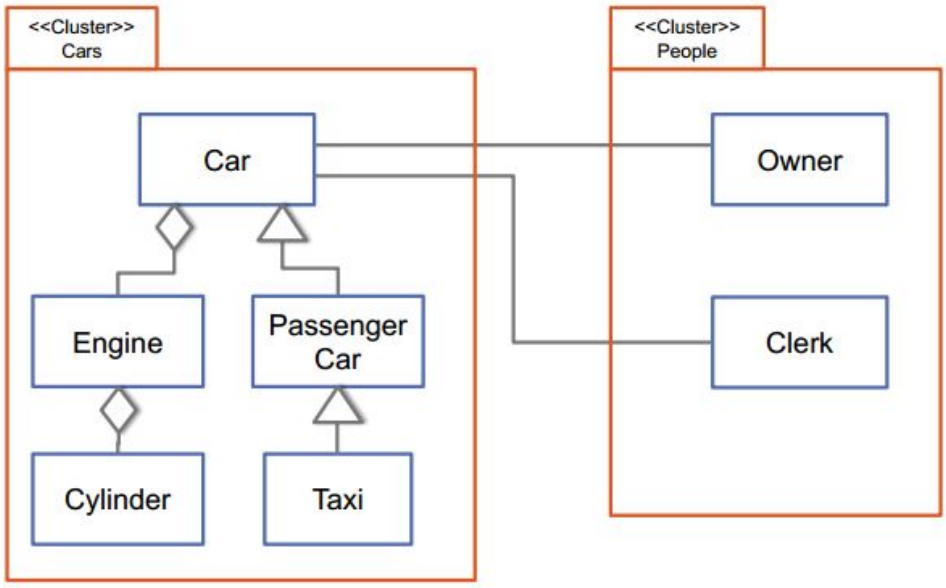
1. Generalization is usually a "is-a" relation
- "a teacher is an employee"

2. Both a taxi and a private car is a passenger car.

3. We want to keep it simple.
- usually there's too much generalizations to make, so don't go crazy

4. Consider inheritance, lower classes inherit from higher classes
- Italic characters indicate an abstract class

Clusters:



1. Clusters is to make our ideas more understandable, we group classes togethr into something moe understandable

2. We often use naming about the main class in our cluster

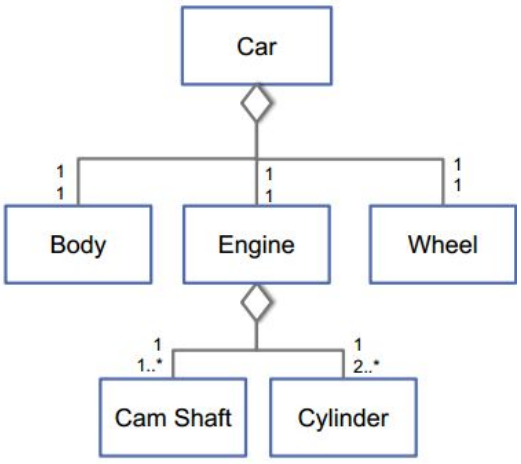
- cluster Cars

3. Similar with a cluster People

- Gather together human actors with a relation to the problem domain

4. We still group them together cus logically they are similar, although not the same

Er en “has-a” / “is-part-of” eller en Aggregation



1. Usually annotates one of three alternatives

2. The picture indicates a whole aggregation

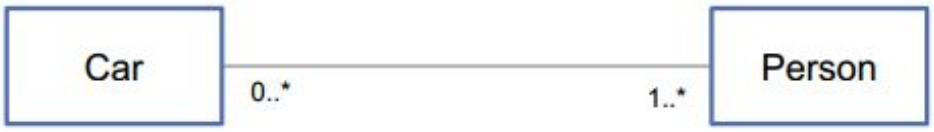
3. If nothing exists in an aggegration it ceases to be a union logically.

4. A union is a group of member classes to a higher master class.
- a master class with no members is a union with no members

5. Car here is a container for its members.

6. The numbers on the picture is to indicate what belongs to what.
- A car can have only one engine, one(4) wheels and one body. It can however have two cylinders.

Er en "knows"/"associated-with" eller Association



1. Meaningful relations between objects
2. This picture indicates a typical relation to an object
- A car is typically in a specific one relation to their owner(person).

3. "Knows" or "associated-with" relations.
- These are objects not classes

EVALUATE SYSTEMATICALLY

Structures must be used correctly

- Generalization vs. aggregation
- Aggregation vs. association

Structures must be conceptually right

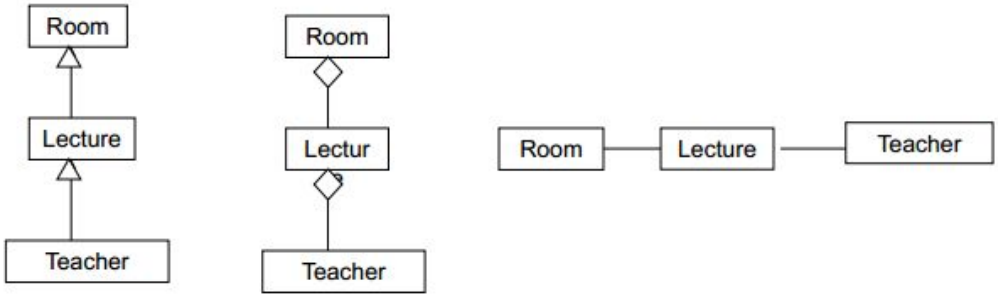
- Names, concepts and structures should correspond to the users' understanding
- The future users

Structures must be simple

- Avoid unnecessary generalizations and aggregations
- Check against the systems definition

Systematically go through this list here when doing this evaluation.

GENERALIZATION, AGGREGATION OR ASSOCIATION?



1. Diamond means Aggregate, Triangle means Generalization, horizontal means association.

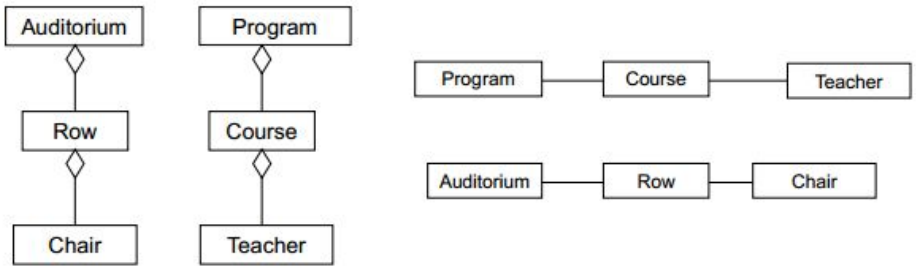
2. The middle one is the correct one, cause you can aggregate rooms to lectures and lectures to teachers.

3. You could also argue that the association template i the most correct one because it's flexible and loose.

- this makes it possible to tailor it easier to our situation.

AGGREGATION OR ASSOCIATION?

- **Can objects exist independently of each other?**
- **Are objects equally ranked?**
- **Can the connection be changed from one pair of objects to another?**
  - The more 'yes' the more it is an association



1. How to determine between aggr. and assoc.
- More yes = more like assoc.

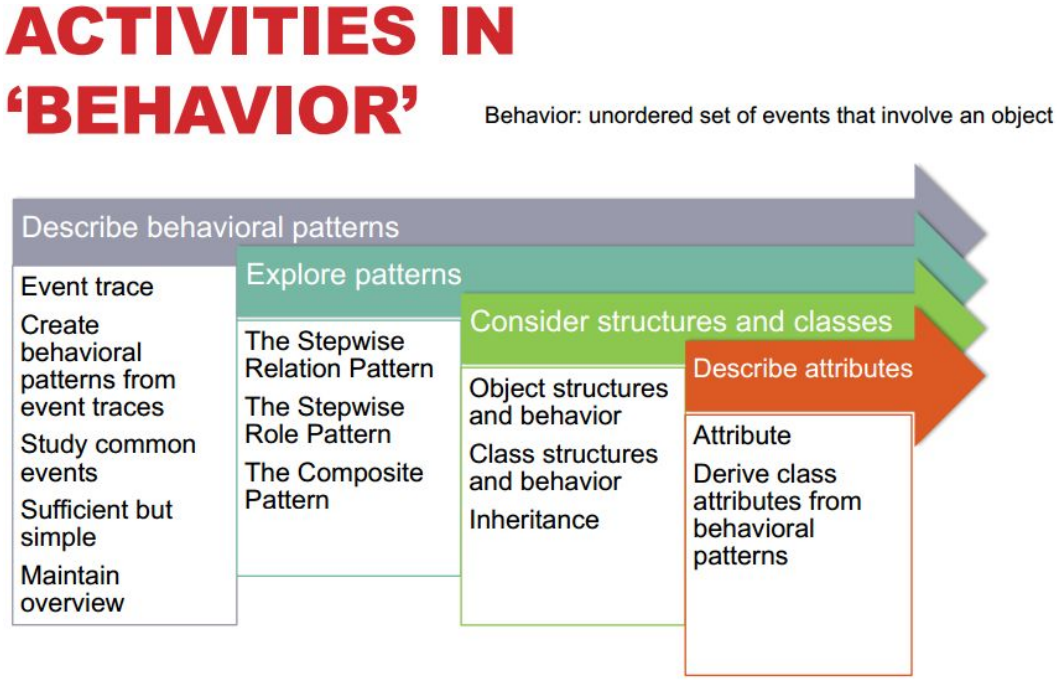
2. If the objects are "equally ranked" meaning do they have same importance.
- Can also look at if something can exist without another object, like Row without Chairs doesnt really make sense in this context.

3. These therefore are not equally ranked.

4. Program, course and teacher are "equally ranked"
- In most cases

5. Some cases although would dictate that theyre not equally ranked
- look at these questions to determine what the situation is for your projekt

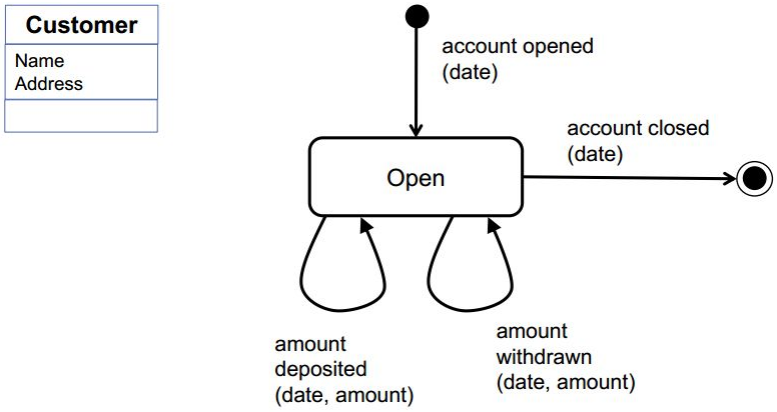
Modeling Behavior



1. Event trace - Try to generate general patterns that the class belong to
2. Sufficient but simple - As little for as much info as possible
3. How do we solve "our" problems.

4. Return to previous phases - We can discover new events and behavior leading to new ideas and classes
5. We derive class attributes from behavioural patterns.

Adfærdsdiagrammer:



1. For each class we will have a description of behavioural patterns.
2. We look at events that create events or terminate events.
3. This diagram doesn't really show us the sequence of events, just the possible events of when the account is open and closed.
4. Customer is here no longer an object

Fra rapport, Adfærdsdiagrammer:

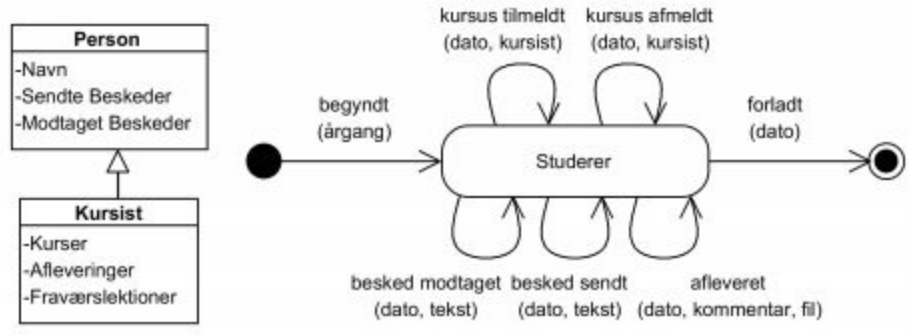


Figure 5.2: Tilstandsdiagram for Kursist klassen

På figur 5.2 kan det ses at *Kursist* klassen arver fra *Person* klassen. Derudover er der tilføjet attributter til de to klasser, der blev udledt gennem analysen af adfærdsmønstrene. Tilstandsdiagrammet illustrerer hvordan *Kursist* klassen samarbejder med hændelserne.

En *Kursist* begynder, herefter kommer man til tilstanden *Studerer*. Under denne tilstand er der forskellige hændelser der kan forekomme. Heriblandt fem iterationer, hvor kursisten kan tilmelde og afmelde sig kurser, modtage og sende beskeder og aflevere afleveringer. Derudover kan kursisten forlade stedet, hvilket betyder at kursisten enten er dimitteret, eller er droppet ud, og derfor terminere den.

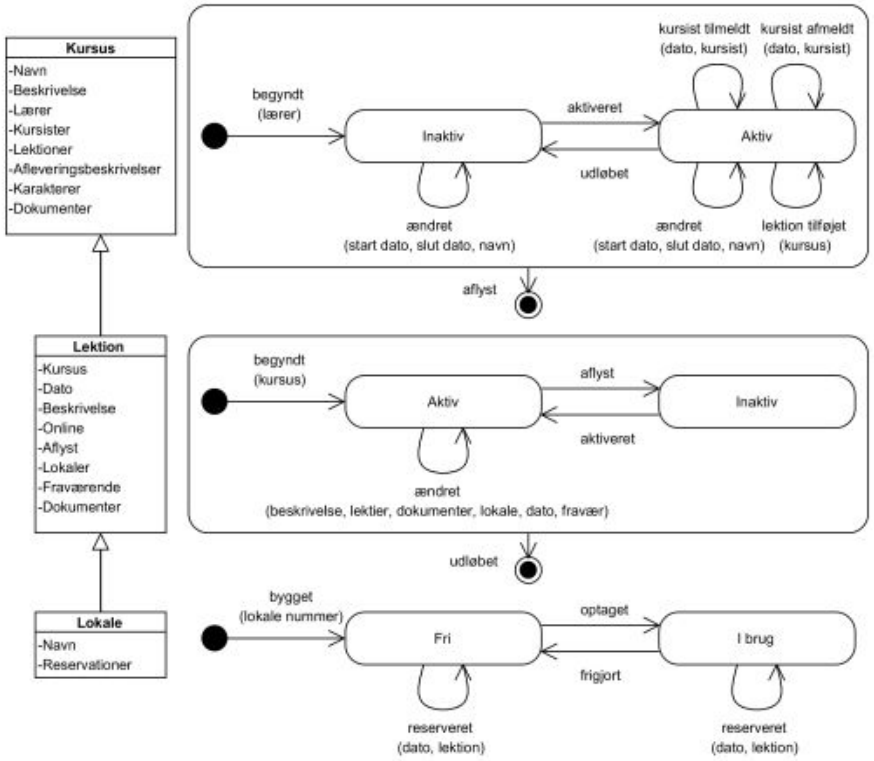
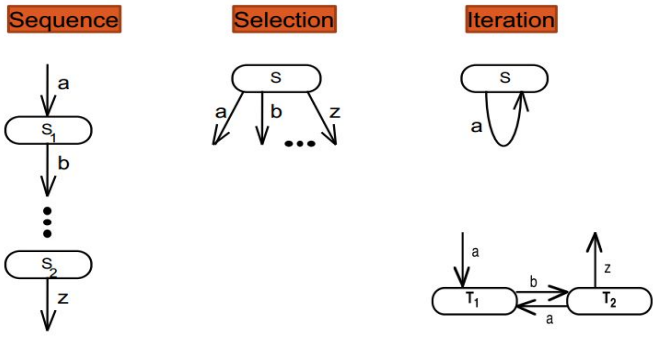


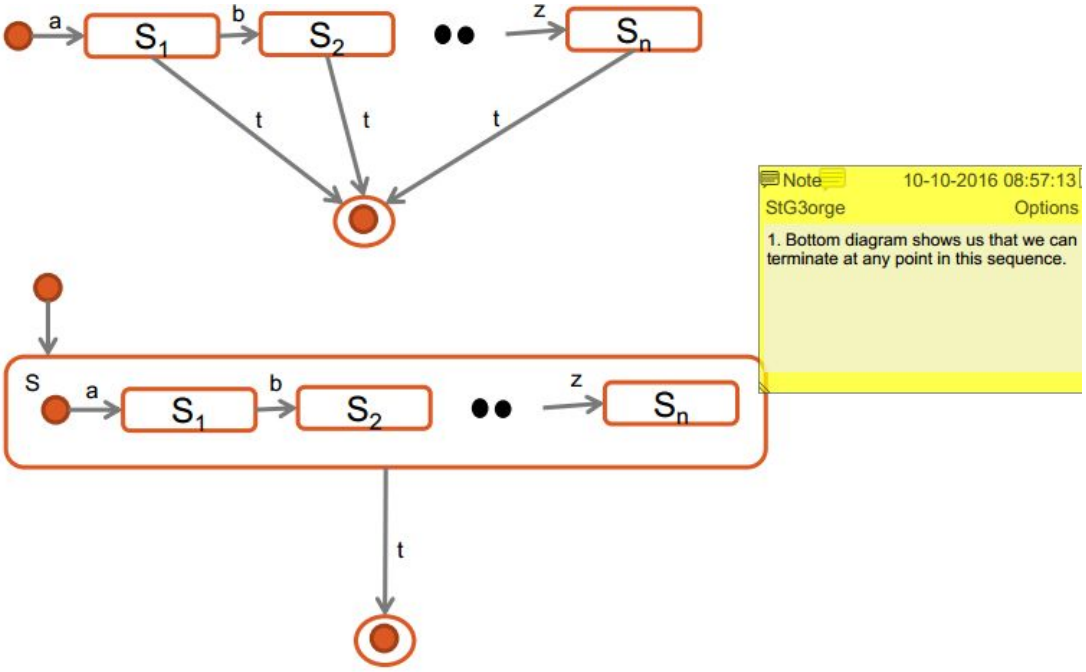
Figure 5.4: Tilstandsdiagrammer for Kursus, Lektion og Lokale klasserne

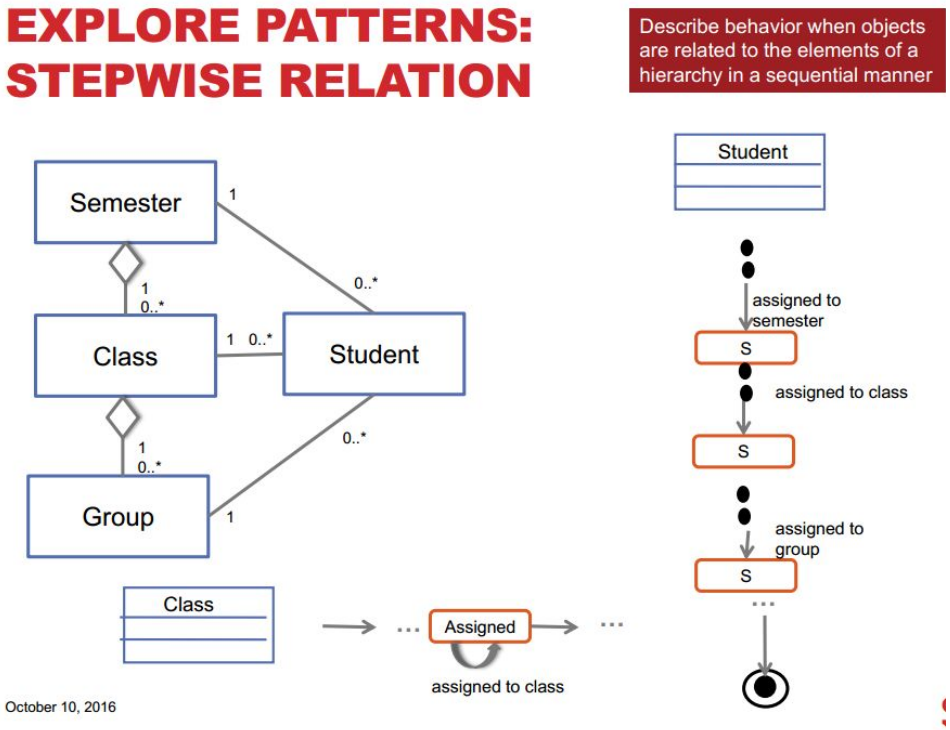
På figur 5.4 vises der tre tilstandsdiagrammer for *Kursus*, *Lektion* og *Lokale*. Samtidig vises aggregeringen mellem de tre klasser, hvor man kan se at *Lektion* aggregere fra *Kursus* og *Lokale* aggregere fra *Lektion*. I Tilstandsdiagrammet for *Kursus* kan man se at et kursus begynder og derefter får tilstanden inaktivt. Dette er gjort, da man ikke vil aktivere et kursus lige så snart det er oprettet, der skal først ændres på kurset og tilføjes lære osv. Herefter kan kurset blive aktiveret, hvor det får tilstanden aktiv. I denne tilstand kan kursister bliver tilmeldt eller afmeldt, samt lektioner kan tilføjes. Derudover kan kurset stadigvæk ændres i den aktive tilstand.

NOTATION: STATECHART DIAGRAMS



Hierarchical states:





1. An Object cannot enter a relatio of another object before another relation is specified.

2. A student must be part of a semester before becoming a part of a class and then part of a group. Group is dependent on Semester and Class.

Application-Domain Analysis

1. Application-Domain er anvendelsesområdet.

Use:

Use cases	Actors			
	Account owner	Creditor	Administrator	Liquidity monitor
Payment	V	V		
Cash withdrawal	V			
Money transfer	V	V	V	
Account information	V		V	V
Credit information		V	V	
Registration			V	
Monitoring			V	
Error correction			V	

Actor: An abstraction of users or other systems that interact with the target system.
Use case: A pattern for interaction between the system and actors in the application domain.

6.1.2 Aktør specifikationer

På figur 6.1, 6.3 og 6.2 er der en aktør specifikationer. Disse består af tre dele formål, karakteristik og eksempler. Formålet beskriver relationen mellem aktøren og systemet. Karakteristikken beskriver de vigtige aspekter af aktørene som bruger systemet. Derudover har vi valgt at supplere specifikationerne med eksempler på hvordan aktørene benytter sig af systemet.

Kursist

Formål En person, som studerer på uddannelsesstedet. Kursisten skal kunne se skema, afleveringer, lektier og lignende informationer vedrørende sit uddannelse.

KarakteristikIfølge spørgeskema afsnit 2.2 ligger kursisterne i aldersgruppen 18-34. Deres gennemsnitlige kompetenceniveau fra en skala fra et til fem ligger på 3,916. Dette betyder at deres erfaring med IT ligger lidt over middel, men samtidig varierer deres niveau.

Eksempler Kursist A er okay med IT og benytter systemet til at tjekke hans skema, aflevere afleveringer og føre fravær. Kursist B er god med IT men benytter de samme funktionaliteter som A men udover det benytter B også systemet til at læse diverse nyheder.

Figure 6.1: Aktør specifikation for kursister

	Aktører		
Brugsmønstre	Kursist	Lærer	Sekretær
upload aflevering	✓		
feedback		✓	
registrering af fravær		✓	
besked	✓	✓	✓
oprette nyhedsopslag		✓	✓
oprette aflevering		✓	
oprette lektioner		✓	✓
oprette kursus			✓
rediger skema		✓	✓
tjek fravær	✓		
tjek skema	✓	✓	✓
tjek nyheder	✓	✓	✓

Table 6.1: Aktørtabel

Sekretær

Formål En person, som arbejder på uddannelsesstedet som sekretær. Sekretæren følger de samme brugsmønstre som en studievejleder, derudover kan de oprette kurser i starten af året og redigere i skemaer.

Karakteristik Ifølge spørgeskema afsnit 2.2 Er sekretærer på VUC&hf 35 år gamle. Samtidig er deres gennemsnitlig kompetenceniveau på 4,5.

Eksempler Sekretær A er erfaren med IT og bruger systemet til eksamensplanlægning, skema samt kommunikation med andre brugere. Sekretær B er også erfaren med IT men har en anden rolle. Han benytter alle funktionaliteterne i systemet jævnligt da han giver support til andre brugere af systemet.

Figure 6.3: Aktør specifikation for Sekretær

Use-Cases(Tilstandsdiagrammer):

1. Hvordan bruger hver bruger systemet

2.Kaldet Use cases.

Fra rapport, Use cases

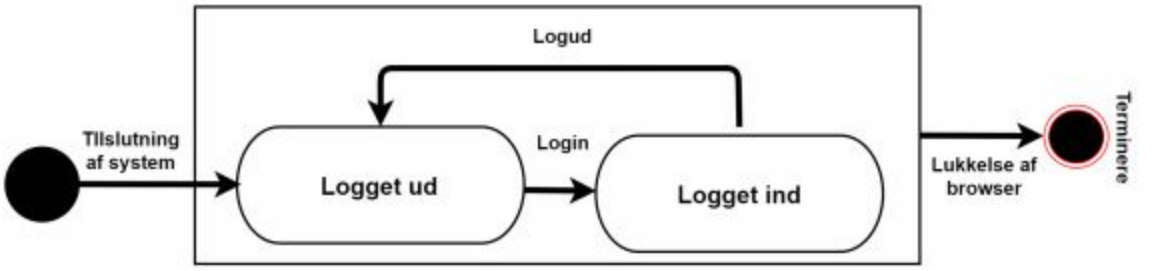


Figure 6.4: Tilstandsdiagram for Login

Login-brugsmønstret er illustreret på figur 6.4. Systemet er en webapplikation der skal reagere på forskellige typer af login alt efter om man er en del af personalet eller en kursist. Man tilsluttes til systemet ved at gå ind på hjemmesiden, hvorfra man bliver præsenteret for en login-anmodning. Man indtaster sit login og derfra er man inde i systemet. Herfra kan man logge ud og ind igen. Terminering sker ved lukkelse af vinduet eller browseren.

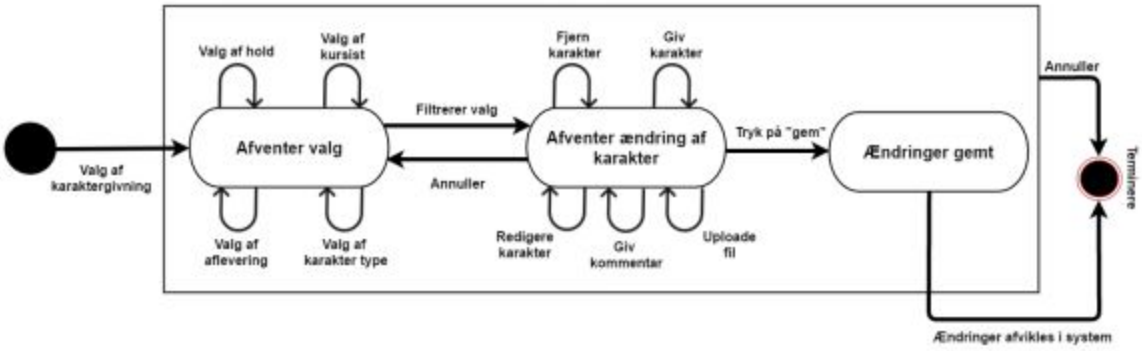


Figure 6.5: Tilstandsdiagram for Karaktergivning

På figur 6.5 vises det at karaktergivning udføres af en lærer, når vedkommende ønsker at angive karakterer for afleveringer, årskarakterer eller lignende. Læreren starter med at angive hvilket hold, kursist, aflevering og type karakter det handler om. Herefter indikerer grænsefladen om at læreren nu kan ændre karakterer. Nu kan læreren uploade filer som besvaring eller give kommentarer samt redigere, fjerne eller give karakterer. Når læreren har fuldført karaktergivningen gemmes besvarelsen.

Objekter Kursus, Lærer, Karakter, Kursist, Aflevering.

Funktioner (Giv karakter, rediger karakter)

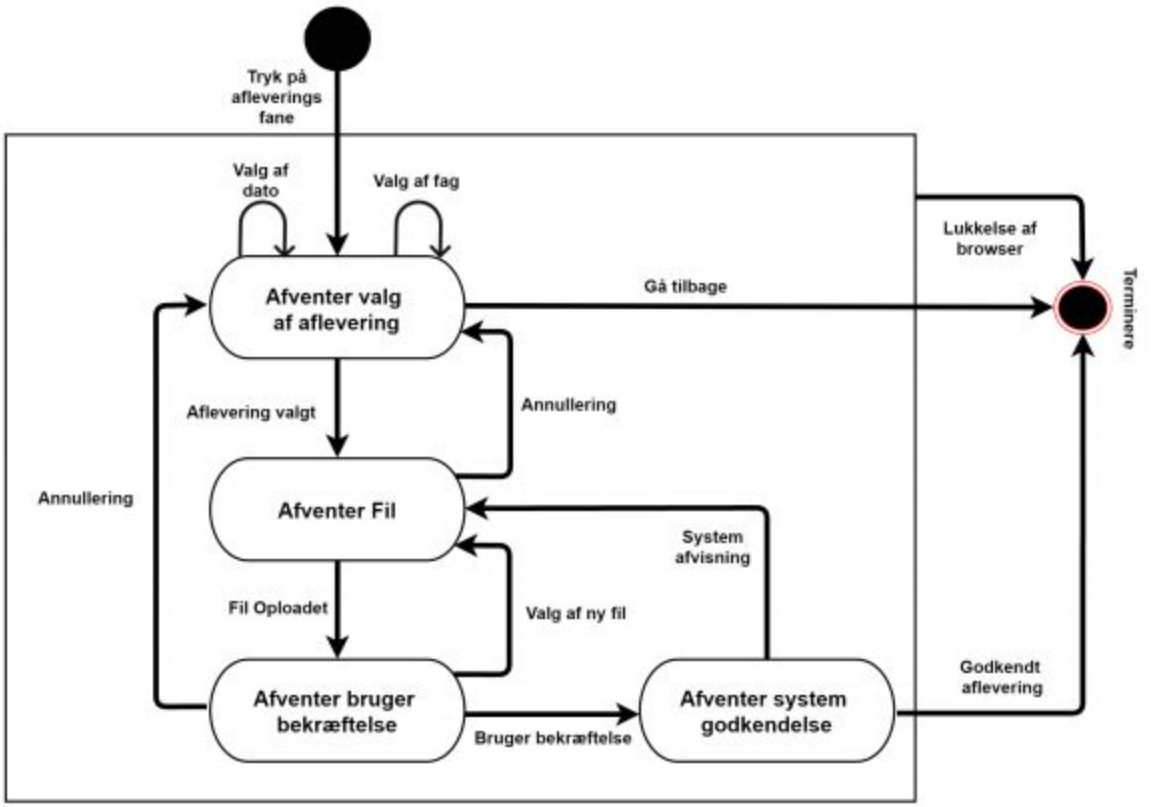


Figure 6.6: Tilstandsdiagram for Aflevering

På figur 6.6 vises det at aflevering igangsættes af kursisten, når vedkommende ønsker at aflevere en opgave tilkoblet et kursus. Kursisten logger ind via sit brugernavn og adgangskode, hvorefter kursisten vælger afleverings-fanen. Kursisten vil nu have mulighed for at vælge en aflevering ud fra en liste af afleveringer, ved valg af aflevering vil afleveringsfasen igangsættes. Kursisten vil få vist et vindue som beder om afleverings-filen, og vil ikke kunne fortsætte processen før en fil er vedhæftet. Kursisten kan vælge at afslutte, eller fortsætte processen ved at trykke på "vælg fil" knappen, hvor kursisten har mulighed for at kigge sine filer på enheden igennem. Når kursisten bekræfter sit valg af fil vil systemet lave en validering af afleveringens-protokollen og enten afvise eller fuldføre afleveringens-processen.

Objekter Kursist, Aflevering.

Funktioner (Upload aflevering, send aflevering, modtag aflevering, download afleveringsdokument)

Application-Domain Functions

1. Evaluate with the users to see if our functions are correct.

2. Have we forgotten a function.

3. Is our list consistent with the previous iterations of our analysis.

Primary result: a complete list of functions

Planning		
Make schedule	Very complex	Update
Calculate schedule consequences	Complex	Signal
Find working hours from previous period	Medium	Read
Enter contents into schedule	Complex	Update
Erase schedule	Simple	Update
…	…	…

1. A list of all functions in the system.
2. Secondary result is a very explained pseudo code-esk outline of a functions intended functionality

Secondary result: specification of complex functions

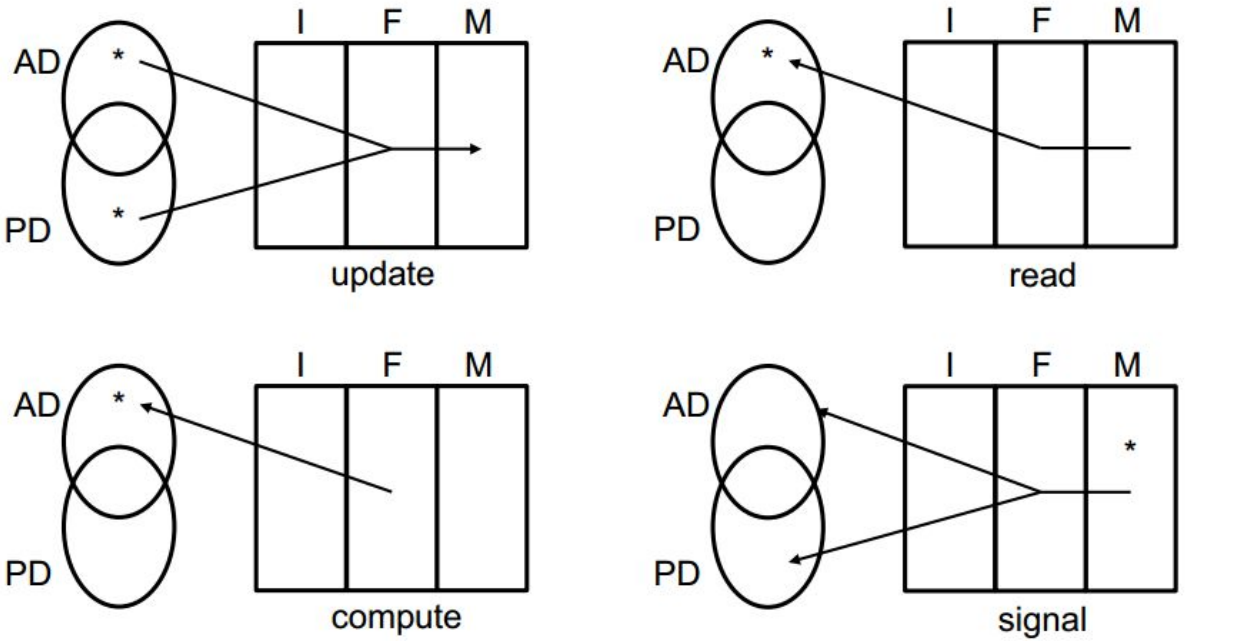
```
Query possible reservations:
        given time or date or employee-name
        search objects in time period-available and select those
                who     belong to emplyee-name, is known
                        have date, if known
                        cover point in time, if known
        result objects of time period-available that fulfill the criteria
```

FUNCTION TYPES

→ Effect of the processing
* Initiative

Function: A facility for making a model useful for actors.

- A resource for actors.



1. The arrows go from a point of origin to an intended destination, such as from both AD and PD to the Model.

2. Read shows something from the model leading to the application domain

3. Compute shows the function leading to the app domain

4. Signal shows that a state in our model should create a change in our AD or PD or both.

5. Figure out what category your function belongs to.

Funktionstabel

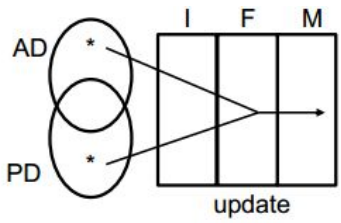
Fra rapport, funktionstabel:

Funktioner	Kompleksitet	Funktionstype
Aflæs besked	Simpel	Aflæsning
Send besked	Medium	Opdatering
Videresend besked	Simpel	Opdatering
Svar besked	Simpel	Opdatering
Aflæs aflevering	Simpel	Aflæsning
Aflever aflevering	Simpel	Opdatering
Hent afleverings dokumenter	Simpel	Aflæsning
Aflæs fravær	Simpel	Aflæsning
Aflæs skema	Simpel	Aflæsning
Hent dokumenter fra skema	Simpel	Aflæsning
Aflæs nyhed	Simpel	Aflæsning
Opret nyhed	Simpel	Opdatering
Slet nyhed	Simpel	Opdatering
Opret brugere	Medium	Opdatering
Vis brugere	Simpel	Aflæsning
Slet brugere	Simpel	Opdatering
Vis kurser	Simpel	Aflæsning
Opret kurser	Medium	Opdatering
Slet kurser	Simpel	Opdatering
Giv fravær	Medium	Opdatering
Aflys afleveringsbeskrivelser	Simpel	Opdatering
Tilmeld kursister	Simpel	Opdatering
Tilmeld lærer	Simpel	Opdatering
Giv karakter aflevering	Simpel	Opdatering
Giv karakter kursus	Simpel	Opdatering

Table 6.2: Funktionsliste, Oversigt over alle vores funktioner

1. Your category can be found in the output and where the information is to be sent to.
2. Our use-cases can help identify problems of all types.
3. This is only to provide overview of all functions.
4. Do not worry about design issues like "what-if"s such as would this function call this other function
5. Outline the functions by intent not by design flaws and functionality. Still in the AD analysis.

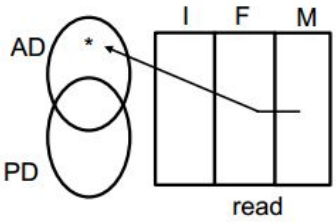
FIND FUNCTIONS: UPDATE



- **Update functions are activated by a problem domain event and result in a change in the model's state.**
- **Questions for each event**
  - How is the event observed, and how is it registered? In which use cases does this happen?
  - How should the use cases be supported by update functions?
  - Which objects, attributes, and object structures are affected by the event, and what requirements does this impose on the update functions?

1. Update functions are activated by a PD event and result in a change in the models state.

FIND FUNCTIONS: READ



- **Read functions are activated by an actor's need for information and result in the system displaying information about the model.**
- **Questions about information needs**
  - Given the work of actors:
    - What do the actors need to know about the state of the model?
    - What read functions does this give rise to?
  - Given the model:
    - Which objects and structures will the actors need information about?
    - What read functions does this give rise to?

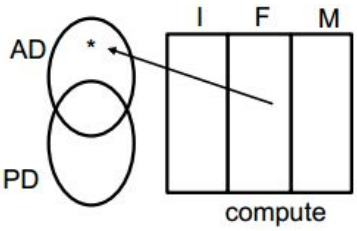
1. Initiated in the AD domain and reflected in the AD

2. What do the actors need to know about the state of the model

3. What read functions does this give rise to?

4. Same for model.

FIND FUNCTIONS: COMPUTE

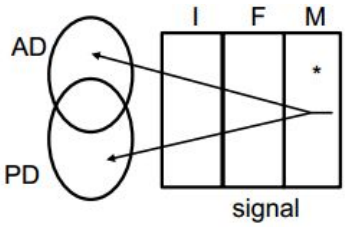


- **Compute functions are activated by a need for information in an actor's work tasks and consists of a computation involving information provided by the actor and/or the model; the result is a display of the computation's result.**
- **Questions about needs for computation**
  - With a starting point in actors and use cases
    - Which computations (not necessarily based on the model) do the actors need to have carried out?
    - Does the computational basis come from the actors, the model, or both?
    - Which computations form complete wholes in the use cases?

1. Initiated in AD reflected in AD

2. Result is a display of the computation result.

FIND FUNCTIONS: SIGNAL



- **Signal functions are activated by a change in the model's state and result in a reaction in the context; this reaction can be a display to the actors in the application-domain or a direct intervention in the problem-domain**
- **Questions about critical states**
  - Examine the model of the problem-domain
    - What are the critical states for the model?
    - What is the significance of these critical states? What are the consequences when they occur?
    - How does a signal function register that the model has entered a critical state?
    - What signal does each critical state give rise to? How reliable and strong do the signals have to be?

Specify complex functions:

- **Mathematical expression:**
  o = f(i)
- **Algorithm**
- **Functional partitioning**

Make schedule	Very complex	Update
• Create six-week period	• Simple	
• Create standard distribution for an employee	• Medium	
• Use standard distribution for en employee	• Simple	
• Adjust the distribution of employees in a week	• Medium	

1. Specify how to show functionality such as a mathematical expression, algorithm or pseudo code for example

2. Partitioning a function can help further understanding. This means parting it up into parts. Can be based on complexity.

Evaluate function-list:

1. Finally we evaluate our list of functions.

2. Is it consistent with our understanding and previous assesments?

3. We can support this with a function prototype, can then go through it with a user.

4. We check our list and check the list until the possibilites are exhausted (find all functions)

5. We might have identified functionality but have an outdated system definition if the functionality is very important and we forgot it.

EVENT, USE CASE AND FUNCTION

What is the difference between an event, a use case and a function?

- **All describe dynamic action**
- **Related**
- **But in different domains**

Event:
What happens to objects in the PD?

Use case:
How will the system be used?

Function:
What is the system going to do?

Example: Order system

- Event
  'Order received' – a customer places an order at a specific time
- Use case
  'Enter order' – a user in the application-domain creates an order for a customer by using the system.
- Function
  'Create order' – an object of the order class is created in the model of the IT-system

1. What is the difference between an event, use case and a function.

2. Events describes what happens to objects in our PD

3. Use cases describe how system will be used.

4. Look at these examples, they differentiate in where they originate.

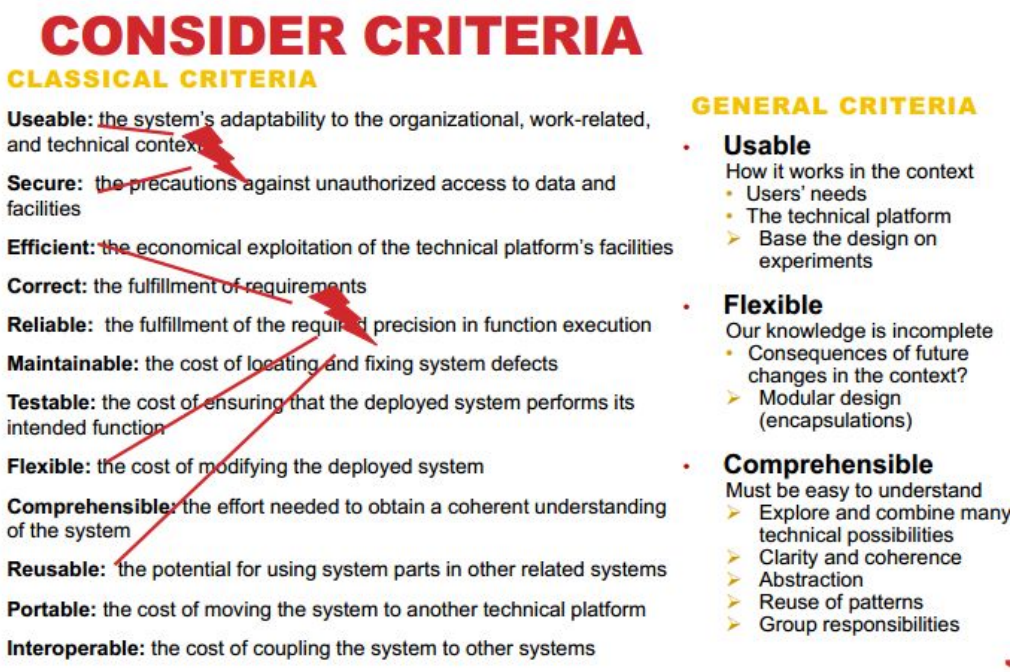
Design: Architechture, Criteria and Components

Fra rapport, kriterier:

Kriterium	Meget vigtigt	Vigtigt	Mindre vigtigt	Irrelevant
Brugbart	✓			
Sikkert			✓	
Effektivt		✓		
Korrekt		✓		
Pålideligt	✓			
Vedligeholdbart			✓	
Testbart			✓	
Fleksibelt				✓
Forståeligt			✓	
Genbrugbart				✓
Flytbart				✓
Integrerbart		✓		

Table 7.1: Prioritering af designkriterier

Brugbart (meget vigtigt) Systemet skal være brugbart, da brugerne anvender det i dagligdagen under forskellige forudsætninger. Derfor skal det kunne benyttes på både computer og smartphones. Det er derudover vigtigt at brugere vil have nemt ved at benytte og lære systemet, selv ud fra forskellige kompetencer og forudsætninger, da platformen vil være et vigtigt redskab for deres studie. Derfor vurderes brugbart til meget vigtigt.
Sikkert (mindre vigtigt) Efter ønske fra informanten om at programmet skulle gemme login informationerne, og hun selv ikke ønskede mere sikkerheds, er dette blevet vurderet til mindre vigtigt. Grundet det ikke er vurderet til irrelevant, er at programmet vil indholde informationer omkring karakter, fravær, afleveringer og beskeder. Derfor kræver det at der er et login til den enkle bruger.
Effektivt (vigtigt) En bruger vil benytte systemet mange gange i løbet af dagen, og derfor er det vigtigt at systemet kører hurtigt, da en bruger vil blive træt af at vente på systemet flere gange i løbet af dagen. Desuden vil mange brugere være tilkoblet systemet på samme tid, og de vil alle have forskellige enheder med forskellige hardware-niveauer. Vi ser det derfor som vigtigt at systemet ikke bruger tid på avanceret visualiseringer som animationer, som vil sænke hastigheden mellem systemet og brugeren, samt at systemet ikke kræver et for højt hardware-niveau.



1. Our knowledge of the future is incomplete

2. We need to shape a modular design by looking at concerns is a way to achieve completeness.

3. System should be easy to understand

4. The General Criteria on the right is a focus area of a proper system criteria listing

Components

Fra rapport, Komponenter

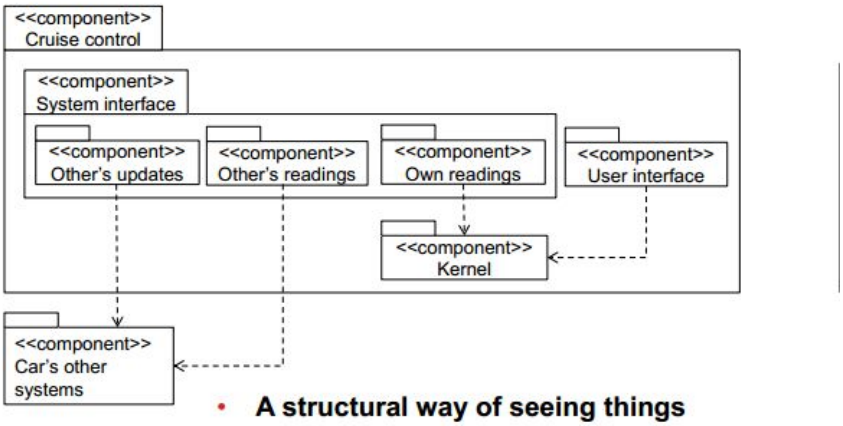
7.2 Komponenter

Vores overordnede komponentarkitektur tager udgangspunkt i mønstret grundarkitektur kombineret med klient-server arkitekturen. Dette er gjort da der bruges et ukendt antal af brugere, som bruger den samme funktionalitet.

Figur 7.1 viser den overordnede komponentarkitektur. komponenten *Server* indeholder komponenter såsom *Brugergrænsefladen*, *Funktioner*, *Model* og den tekniske platform. Dataen skal være tilgængeligt for brugerne og derfor er alle komponenterne samlet på en server.

Klienten kobler sig til serveren via brugergrænsefladen i programmet og hvis der sker ændringer i brugergrænsefladen, vil dette få en effekt på *Browser* klienten. Samtidig er brugergrænsefladen relateret til funktioner, da funktionerne kan blive rekonstrueret, slettet eller oprettes nye, det er netop funktionerne som bestemmer hvordan brugergrænsefladen ser ud og hvordan den opføre sig. Derudover er *brugergrænsefladen* afhængig af komponentet *Bootstrap*, da bootstrap bestemmer udseendet af brugergrænsefladen. *Model* kan give en effekt til funktonerne, da modellen inderholder alle klasserne, og hvis de ændre sig så opføre funktionerne sig anerledes. *Model* er afhængig af komponentet database, da alt dataen bliver tilført lister i modellen, således disse kan tilgås i *Funktioner*.

RESULT OF 'COMPONENTS'

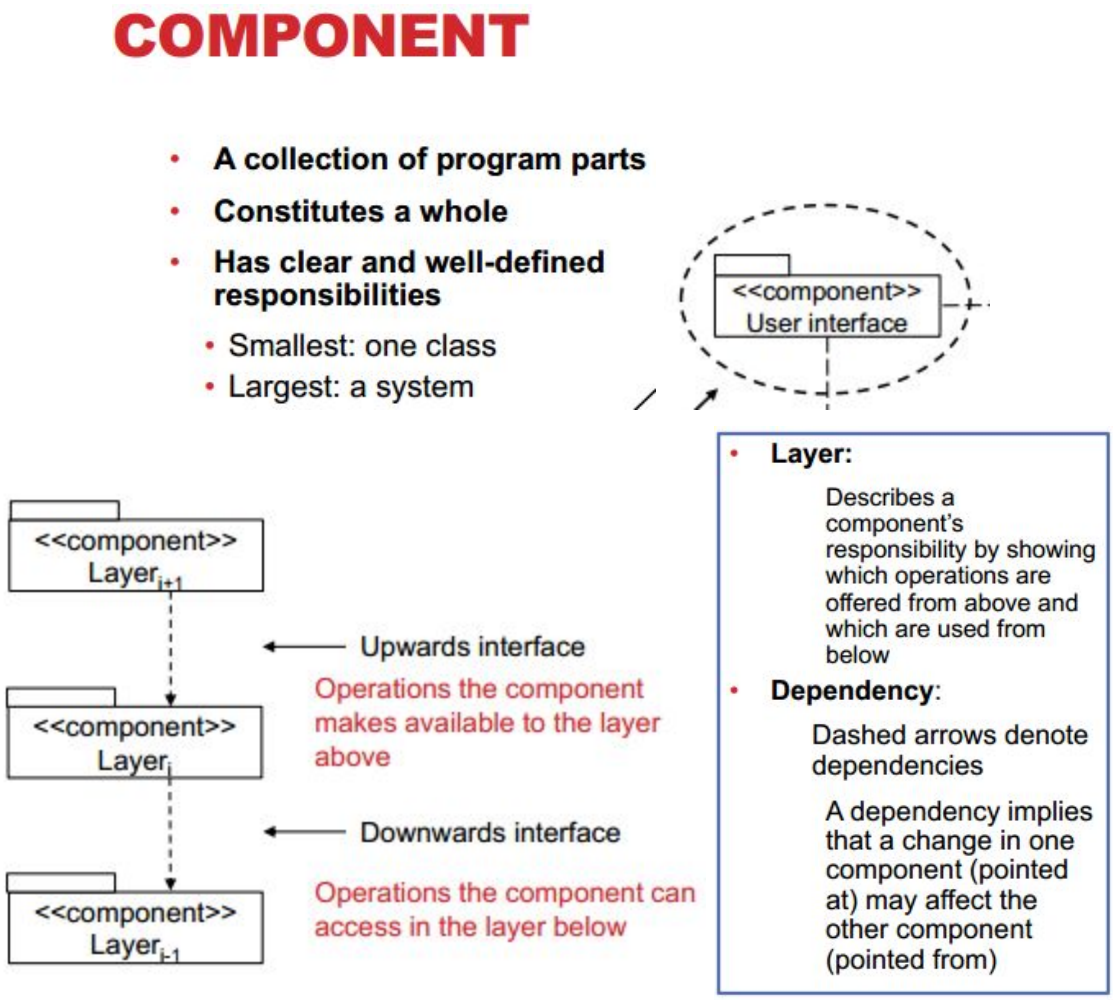


- **A structural way of seeing things**
- **Divides the concerns of a system**
- **Directs attention to comprehensiveness and flexibility**

October 19, 2015

1. This is a way of dividing components into a comprehensive diagram.
2. All systems should be modeled like this

1. A component is a part of the system responsible for a specific purpose or function.
2. The smallest component we have is a class.
3. The largest is the whole system.
4. We want to be somewhere in between to comprehend our architechture.
5. We can add notes to a component to specify its purpose or interaction

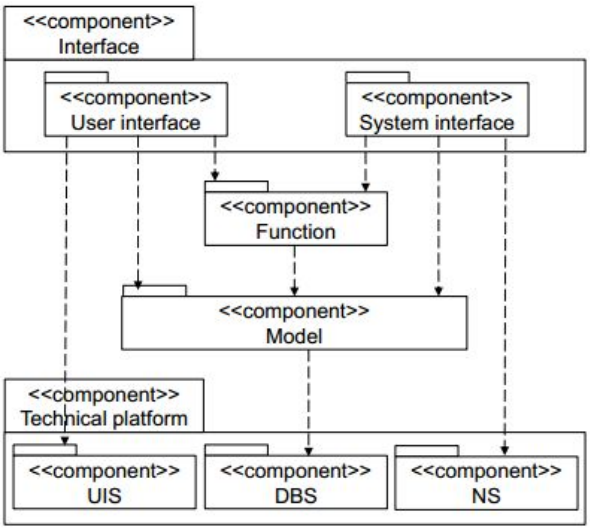


1. Which is used from above and what is also used from below.

2. Thus changing one layer may affect the next layer

3. Operations coming beneath a top layer of a component is a way to structure components and their functionality in tandem with a greater use/function of the system.

PATTERN: GENERIC ARCHITECTURE



- The generic architecture reflects the division between the problem domain and application domain
  - **Model component**: implements the problem-domain model
  - **Function layer**: contains system functions
  - **Interface**: can be decomposed into two separate parts: UI and SI
- "The technical platform" is an extension and encapsulation of the underlying technical platform

Fra rapport, komponentdiagram:

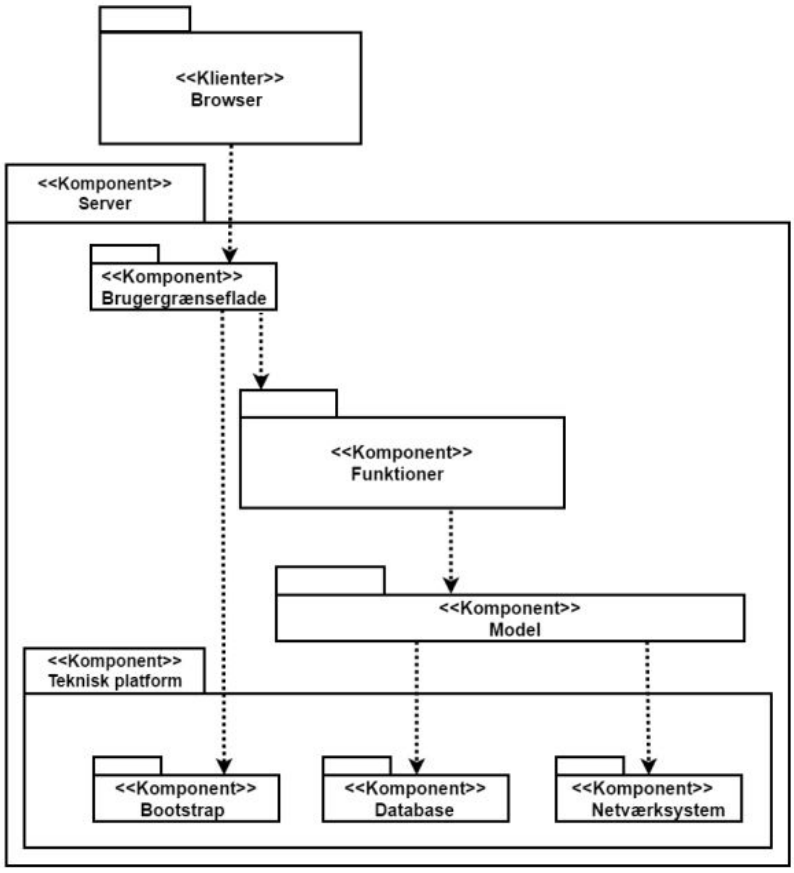
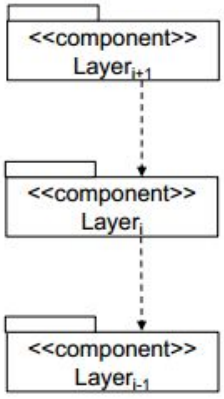


Figure 7.1: Overordnede grundarkitektur

VARIATIONS



- **Closed architecture:** can only use operations from the layer immediately above or below
- **Open architecture:** can use operations from any layer above or below
- **Strict architecture:** can only use operations in layers below
- **Relaxed architecture:** can use operations in layers below and above

Closed-strict	Open-strict
Closed-relaxed	Open-relaxed

Many layers a closed-strict may be best
Few layers an open-strict may be best

1. By associating some components with others we can accidentally violate OOP rules of programming

2. Open architecture is the epitome of this possiblity.

3. List a component and its layers what most seems fit

VARIATIONS OF THE CLIENT SERVER ARCHITECTURE

- Based on the client server and the general patterns
- Five component architectures with varying degrees of distribution

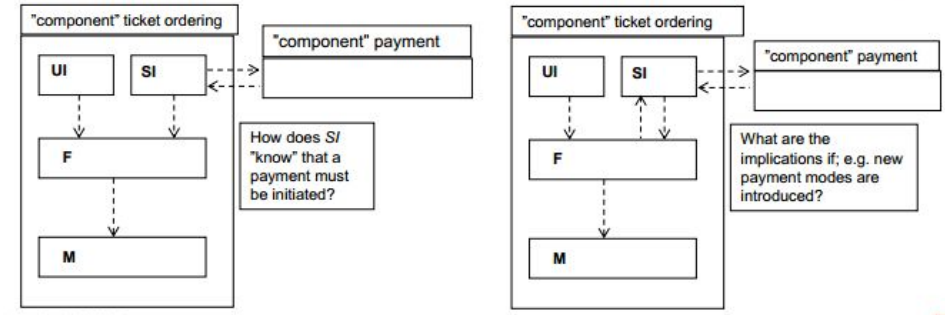
Klient	Server	Arkitektur
I	I + F+ M	Distributed presentation
I	F + M	Local presentation
I + F	F + M	Distributed functionality
I + F	M	Centralized data
I + F + M	M	Distributed data

1. A thin client is just the interface, no substance just an interface

2. A thick client is a client with functionality and data int he client. The information on the client can communicate with the communicator.

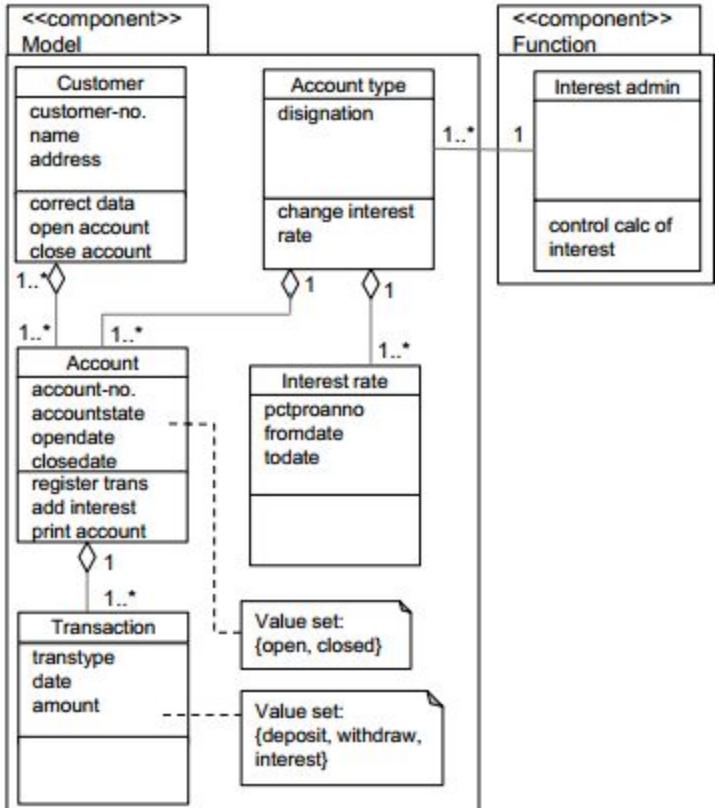
DEFINE SUBSYSTEMS

- Large systems must be decomposed (several independent subsystems)
- Each subsystem has its own architecture
  - Based on the generic architecture pattern
- A subsystem's "System interface" component provides other sub systems with a coherent interface for accessing the given subsystem's functionality



Alternatives

Model-Component



1. Stay true to the architecture initially and only change it if absolutely neccessary and new things are discovered.

2. Components are a block of the program doing a specific task, connecting these in a framework is a component design.

Fra rapport, Modelkomponent diagram

På baggrund af vores problemområde har vi udarbejdet en modelkomponent for systemet. Der er blevet rekonstrueret det tidligere klassediagram se afsnit 5.2 og vurderet hændelser og klasser fra problemområdet. Dette er gjort for at finde ud af om der er opstået nye klasser eller om hændelserne som kan ændres til klasser.

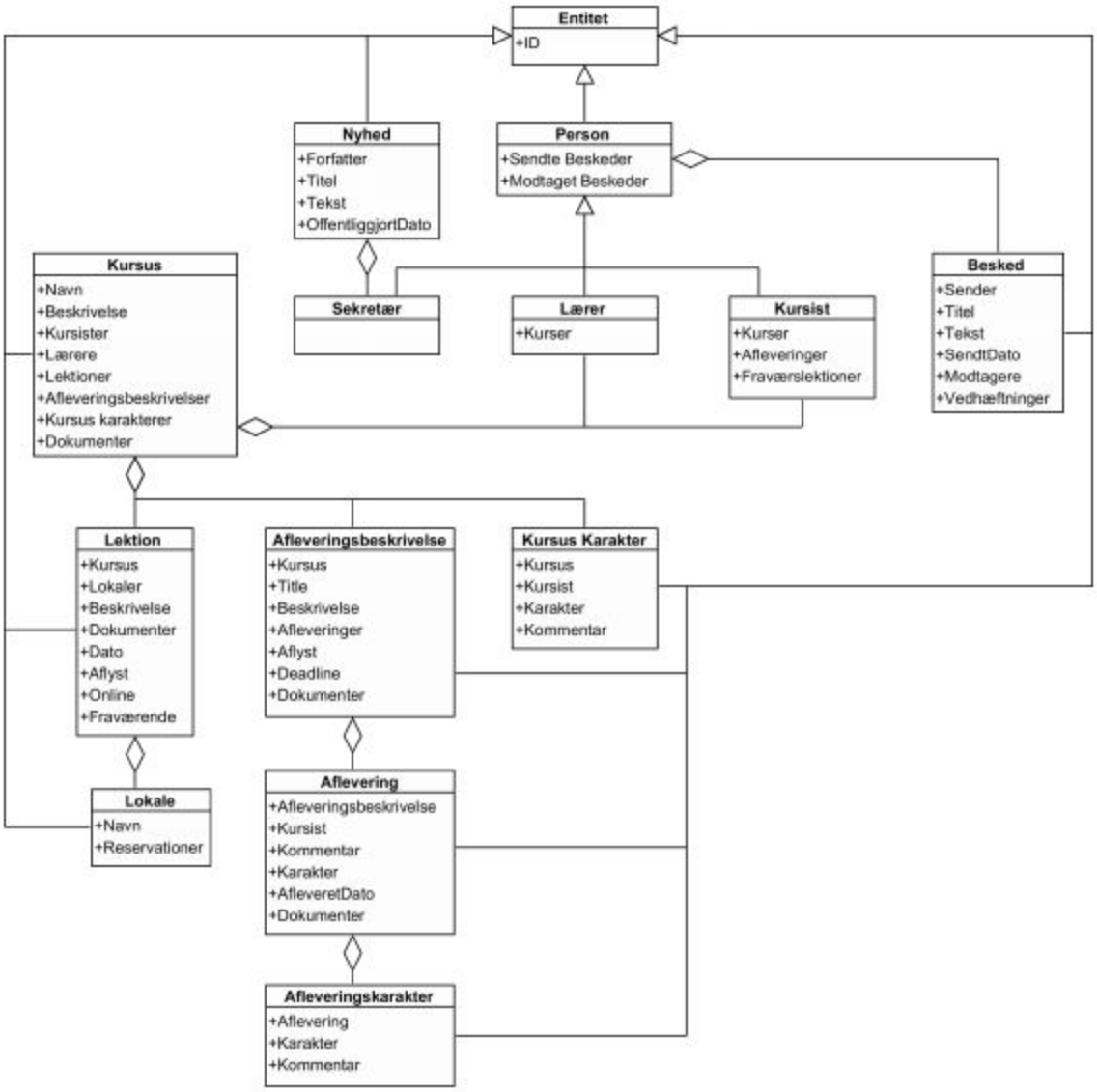
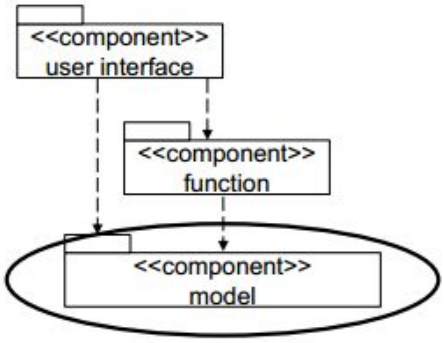
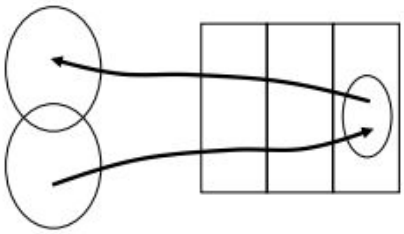


Figure 7.2: Klassediagram for modelkomponent

Figur 7.2 viser det rekonstrueret klassediagram, som er vores modelkomponent. Dette består af 14 klasser, hvoraf de 13 klasser arver fra *Entitet*. Der er gjort dette, da alle klasserne skal bruge et id, hvilket er generet af vores database og disse bruges til associeringer mellem klasserne. Derudover er der tilføjes attributter til alle klasserne, således man kan se hvad de indeholder.

- **Component:**
  A collection of program parts that constitutes a whole and has well-defined responsibilities.
- **Responsibility of the model component:**
  maintain an updated representation of the problem domain.

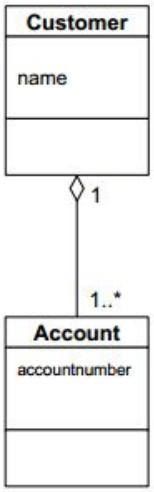


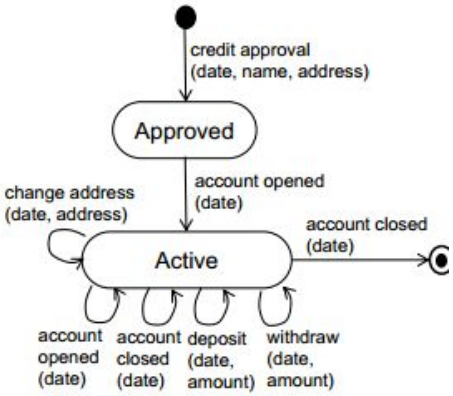


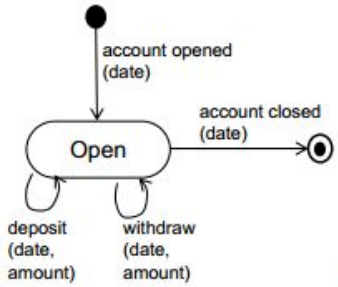
ANALYSIS MODEL FOR BANK SYSTEM

- **Class diagram**
- **Event table**

Event	Customer	Account
Credit approval	+	
Change adress	*	
Account opened	*	+
Account closed	*	+
Deposit	*	*
Withdraw	*	*







REPRESENT PRIVATE EVENTS

events that involve only one problem-domain object.

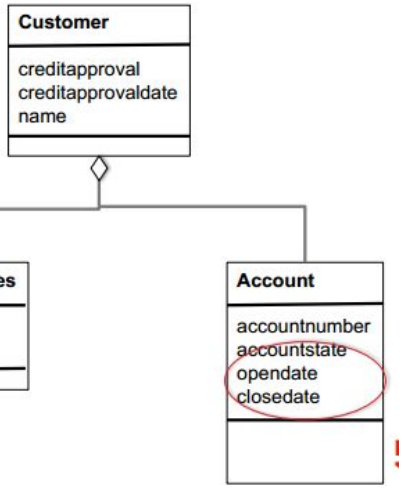
- **Sequence and selection**
  - Represent these events as an attribute in the class described in the state chart diagram.
  - The system assigns a value to the attribute when the event occurs.
  - Integrate the attributes of the event in the class.
- **Iteration**
  - Represent these events as a new class, connect it to the class described in the state chart diagram with an aggregation structure.
  - The system generates a new object of the class each time the event occurs
  - Integrate the attributes of the event in the class.

1. Reason for having iteration as their own custom classes and not as an attribute of a class means we can easily keep track of old data.

REPRESENT COMMON EVENTS: CHOOSING A SIMPLE ALTERNATIVE

- **The events 'account opened' and 'account closed' are interative on class Customer and in a sequence on class Account**
- **The simplest representation is by adding attributes to class Account**

Eventç	Customer	Account
Credit approval	+	
Change adress	*	
Account opened	*	+
Account closed	*	+
Deposit	*	*
Withdraw	*	*



October 21, 2015
SU:L10:E15

1. Account is sequential, customeraddress is iterative.
2. Account open and close are iterative on customer and a sequence of class account.

Fra rapport, Funktionskomponent:

7.2.2 Funktion komponent

I dette afsnit vil funktionskomponentet blive designet med udgangspunkt i funktionerne fra afsnit 6.3 anvendelsesområdet. Der vil være beskrivelser af funktionerne i forhold til hvordan de skal designes. Disse beskrivelser bruges herefter til implementeringen af funktionerne. Der er taget udgangspunkt i en række funktioner, disse funktioner er valgt da de er prioriteret højere end de andre funktioner. Disse er prioriteret højere, da det var de centrale funktioner som informanten ønskede, derudover er de også prioriteret ved en analysering af spørgeskemaet. Funktionerne er listet med beskrivelser nedenfor.

Aflæs aflevering Denne funktion har til formål at kursister kan aflæse deres afleveringer. Dette gøres ved at attributterne fra afleverings klassen opstilles systematisk i en tabel, således der skabes et overblik over de forskellige afleveringer. Samtidig skal den opdatere om den er afleveret eller ikke afleveret.

Aflever aflevering Da en af attributterne på aflevering indeholder en filepath, så skal denne tages i brug her. Funktionen skal sørger for at der kan uploades et dokument så det kan downloades igen af andre brugere. Denne opdatere modellen ved at gemme en ny aflevering.

Hent afleveringsdokumenter Denne funktion har til formal at downloade afleveringsdokumenter, og dette gøres ved brug af en filepath. Måden man henter dokumenterne er ved brug af afleveringstabellen, herefter skal den finde den bestemte filepath til afleveringsdokumentet.

Fra rapport, Design principper:

8.1 Design principper

Vi har valgt at følge nogle design principper der beskrives af Benyon[1, s. 88]. Disse principper vil generelt sikre sig kvalitet og brugervenlighed af brugergrænsefladen. Der er lavet en liste over design principperne, i næste afsnit vil disse blive anvendt.

1. **Visibility** Det skal være let for brugeren at se hvad systemet foretager sig. Systemet skal være genkendeligt, da man nemmere vil genkende frem for at huske. Samtidig skal systemet sættes op, således layoutet af information kan tilpasse alle skærmstørrelser, på denne måde øger man muligheden for visibility.

2. **Familiarity** Der skal sørges for at sprog og symboler er noget som brugeren allerede kan relatere til i forvejen. Samtidig skal der være brug af metaforer, da brugeren kan relatere til disse fra andre lignende systemer og hjemmesider.

3. **Recovery** Systemet skal indeholde *"error recovery"* hvilket betyder at der skal forekomme advarsels-signaler, så som *"Er du sikker på du vil slette denne kursist"*.

4. **Affordance** Designet skal være genkendeligt for brugeren. Dette betyder systemet kan knyttes til andre systemer, hvilket brugeren har kendskab til. Samtidig skal der gøres brug af knapper, tekstbokse og navigationsbar af samme grund.

5. **Navigation** Der skal være simpel navigering. Dette kan gøres ved brug af en navigationsbar som menu. Her kan brugeren bruge denne til at vælge