# Application-domain analysis - Use

Until now, he hasn’t started giving feedback on our assignments.

Keep an eye on moodle, as the next assignment will become public during it.

## The assignment

We had a system definition and an interview.

There is an example of an event table in the first slides.

We need the last “Price group created/cancelled” and “station created/closed” because while they’re not a direct part of the system definition, they are events regarding objects we need to monitor and administrate.

See that Esben has utilized both the hierarchy pattern (Station, price group, car) and the role pattern (Contract, reservation, rental).

Note that the Customer class should be an abstract class.

Question: Could you argue that price group could be an attribute?

Answer: Maybe, when implementing, we could do that. But right now, we are trying to model it correctly to get a good understanding of the problem domain. The price group is a pretty big part of the system definition and problem domain, so we need to include it.

## Application-domain analysis

## Model the context

The context is the problem domain and application domain. Remember that the application domain is the organization that administrates, monitors or controls a problem domain.

## Emphasize the architecture

When we did the problem domain analysis, we gained a lot information about the model part of the MCV system. Now, we’re going to learn a lot about the interface components.

## Application-domain analysis - result

We’re going to talk about use today, functions later, and interfaces in the end. These are the three activities in the application-domain analysis.

So, we’re going to look at how the system interacts with people and systems in the context.

The divisions between problem and application domain is that it gives us insight into some other aspects than problem domain would. The model resulting from the problem domain analysis is likely to not change a lot. Interface properties, however, which is what we’re going to find here, is going to be subject to change a lot more than the model. Furthermore, changes in the interface do not require changes in the model, while the vice versa situation isn’t always true.

## Purpose of application-domain

To determine a systems usage requirements.

The output should be a complete list of the systems overall usage requirements.

## Use

The purpose is to determine how actors will interact with a system.

An actor is an abstraction of users or other systems that interact with ours.

A use case is a pattern for interaction between the system and actors in the application domain.

We determine the application domain with the use cases, and evaluate use cases in collaboration with users.

The outcome should be descriptions of all use cases.

## Result(1): Actor and use case overview

The actor table. Looks like an event table, but is different.

The main difference is that the elements in the column to the left are no longer events.

## Result(2): Actor and use case specifications

A description of the actor (see the slide), and a concise description of a use case relevant to the actor.

Some of the activites we’re going to do here is to find actors and usecases, and we do this by analyzing tasks. We then identify the actors, describe them, same thing with the use cases. Then we structure the actors and use cases with each other. We then explore for patterns, and then evaluate the actors and use cases for consistency (are they related to the same task? Logically sound?). We can also use prototypes to evaluate our actors and use cases.

## Analyzing tasks

We determine the application-domain with use cases. We avoid detailed descriptions of “how it used to be”.

The goal is to get an overview of the work tasks in the application domain, and especially the division of work and the task boundaries.

What are our information sources? Mostly the system definition, we have some rules and procedures from textbooks on how to do things, we can observe people, interview, participate in the work with users, and so on.

## Example

When describing tasks, give them:

* Name and contents
* Purpose
* How are the tasks assigned?
* Who performs the task?
* Relation to other tasks
* Result

See the slide for an example, I’m not fast enough for this.

## Find and describe actors

When identifying actors, we determine the division of work and the task related roles in the target systems context. Different users have different roles, so our users/actors do not relate to the single individual, but more the role itself.

When describing actors, give them a goal, characteristics (aspects of the actors use of the system) and an example (general characteristics).

See the slide for an example of an actor description. Those are examples of two very different actors that are both interacting with the system, but both in the role of the account owner.

## Identify use cases

When we identify use cases we describe how actos interact with the system to complete their tasks. We try to minimize the overlap between use cases. We also try to make them coherent, so that the actor relates to the use case.

We write scenarios with examples of how to complete the tasks. We do this with text and state chart diagrams.

See the slide “Describe use cases” for two examples of how to describe a use case.

You see both a description in the form of a textual scenario, and a state-chart diagram.

In addition to the text, we can add the objects and functions required to support the use case. Note that the words at the arrows between states are not events, but interactions. For example “prompt for code” is an interaction between actor and system, not an immediate event.

## Example: Bank

Another example, not the ATM system, but the system with account owners, and so on. Note that “liquidity monitors” are not people, but is another system that interacts with this system.

We can also relate use cases to the actos graphically in the slide “Use case overview(1): Use case diagram”. It should be self-explanatory. Another way of illustrating this is to make an actor table like the one we saw before. We have a list of actors and a list of use cases and the checkmarks that associate the actors with the use cases.

We can also cluster the use cases within an overall focus and model which of the actors are related to the cluster(s) or to single use cases within the clusters.

## Patterns

We have patterns that we try to reuse when looking at use cases.

**The procedural pattern**

We have some states that can only be entered given that some interactions are executed in a specific procedure. This pattern is useful when we want to ensure that business rules are being followed. That can be very efficient.

**The material pattern**

Here, the actor can do almost everything in any order.

The generic way of showing this is you have one general state, and iterative actions on that state. We can also have indirect iterations where one action can lead to a different state, and return to the general state when another action sends it back.

## Similarities and differences between use cases, actors, events and objects

See the slide for a comprehensive comparison.

Actors and classes describe the more static attributes of the domain we’re trying to model.

Use cases and events are more dynamic in nature, and describing the dynamics in the domain we’re trying to model.

## Evaluate systematically

Each use case should be simple and constitute a coherent whole.

Description of actors and use cases should provide understanding and overview of the application domain.

Use cases should be described in enough detail to enable identification of functions and interface elements.

Test patterns of use with users.

We mentioned that prototypes can be used to test use cases. See the slide for the different acitives when testing use cases with prototypes.

## Evaluate social changes

What are the changes in the application domain when our system takes action?

When we are trying to identify social changes, we can look at work content, autonomy and control, social relations, and education and development.

In the mechanistic extreme, the work content may be specialized jobs, and many things that are being regulated. Autonomy might be monitoring, the stressful load of employees, and so on. The social relations may include less security, the users may become alienated from the system, not a lot of social interaction and so on.

The romantic extreme differs.

In the work content, consequences might by no division of labor, no procedures and rules, and everything is regulated by consequences.

*This is going way too fast to write down, look at the slide.*

It’s important to understand this part of the evaluation, as it’s pretty important to understand the social changes your model of the application domain might lead to.

# Exercises

Se SW3\_Project\_Temp.pdf der ligger i denne lektions mappe. Se afsnittet “Application Domain” for at se hvordan vi har lavet vores application domain analysis.

## Opgave 15

**Aktør**: Mobilhaver

**Formål**: At kommunikere med venner, familie og bekendte vba. sin mobiltelefon når face-to-face kommunikation ikke kan lade sig gøre.

Karakteristika: Systemet omfatter mange forskellige brugere med unik information og varierende teknisk erfaring.

**Eksempler**:

1. Mobilbruger A har altid haft problemer med at huske navne og telefonnumre. Han har ikke meget forstand på teknologi, og har behov for at interaktion mellem ham og telefonen er så begrænset som muligt, da han helst foretrækker face-to-face kommunikation.
2. Mobilbruger B er teknisk interesseret i al teknik hun rører ved, og prøver derfor sin telefon af for funktioner og muligheder. Mobilbruger B agter at bruge sin telefon så meget som muligt, og prøver for det meste at kommunikere igennem denne frem for face-to-face kommunikation.

**Brugsmønstre**:

1. Anne-Lotte skal ringe til hendes mor. Hun kan desværre ikke huske hendes mors nummer, og kan heller ikke huske om hun har gemt det tidligere. Hun prøver at søge efter sin mors navn i kontaktbogen, og finder en gemt kontakt under hendes mors navn.
2. Hans-Erik snakker sammen med en kollega efter arbejdet. Hans-Erik har altid været dårlig til at huske navne, og dermed også telefonnumre. De aftaler at mødes senere den weekend, og for ikke at brænde sin kollega af på aftalen senere gemmer Hans-Erik kollegaens nummer i sin kontaktbog på telefonen.
3. Torben-Arkibal er blevet ringet op af en gammel ven for to dage siden. Han glemte at fortælle ham noget vigtigt dengang, og har brug for at få fat på ham igen. Han kan desværre ikke huske hans nummer, da det var hans ven der ringede til ham. Torben-Arkibal kan dog huske hvornår de ca. snakkede sammen. Han går ind i sin samtalelog og finder deres samtale, hvori han finder sin vens nummer.

**Aktørtabel**

Da vi kun har én aktør bliver de bare en liste af brugsmønstre der er koblet til aktøren.

*Mobilhaver*

* Ring
* Læg på
* Tag telefon
* Opret kontakt
* Slet kontakt
* Find kontakt
* Find samtale

## Opgave 6.17

### Aktører

**Passager**

**Formål:** At komme fra en etage til en anden etage vba. elevatoren.

**Karakteristika:** Mange forskellige personer der skal forskellige steder hen, måske med varierende teknisk erfaring.

**Eksempel:** Passager A er en gammel dame der ikke har meget forstand på noget teknologi. Behøver et intuitivt og simpelt interaktionssystem.

Passager B har meget travlt når han skal frem og tilbage i bygningen, og kræver derfor en hurtig respons i systemet.

**Elevator controller**

**Formål:** At sende elevatoren til den forespurgte etage.

**Karakteristika:** Der er én af dem i hvert elevator, og interageres med via et panel af knapper.

**Eksempel:** Controller A er en computer der sidder i elevatoren, og modtager passagerernes input. Controlleren udfører derefter den logik der skal til for at få elevatoren fra A til B.