# System Choice

## The system context

As we’ve gone over before, when designing a system we’re concerned with two main domains; the problem domain and the application domain. The problem domain is the part of a context that needs to be administered, monitored or controlled by a system, and the application domain is the organization that administers, monitors or controls a problem domain.

We use the domains to interpret a problematic domain, wherever that may be.

However, there are many ways to interpret the problematic situation. So how do we choose? We collaborate with the future users!

For reference, “System choice” is about defining the system we’re going to work with

## System Choice: Three Sub Activities

The activity of defining a system is usually split into three subactivities:

**Situation**

This is where we aim to understand the situation and how stakeholders (the users) see it.

A helpful tool for understanding a situation is to create an overview of the situation with “Rich Pictures”. Read about these in the book, they’re basically drawings that, for example, represent how the work flows in an office; they represent how the drawer sees the situation as it is. They give a clear vision of the inner workings of the problem and application domain and makes it easier for the systems designer to evaluate what changes need to be made.

**Ideas**

This is where we criticize existing traditions. It’s important to keep the “skeptic” mindset during this activity, as you need to question everything that is done traditionally, so that we can catalyze the process of thinking of new ideas and new ways to operate. This is usually also where new ideas are tested and tried using prototypes.

First, we look at existing solutions. You can visit other organizations or study standard systems on the market to get an idea of what the traditional systems are made of.

When looking at existing IT-solutions, answer the following questions:

* Which ideas lay the groundwork for the system?
* Do the ideas seem useful? Why?
* Will the ideas work in your context? Why?
* Can the ideas be adapted to your system? How?

You can also use metaphors to describe the system in a different way. This helps transfer ideas and experiences from one person with a set of skills to another person with another set of skills. For example, you can look at a library system as an inventory control system instead for the purpose of making your fellow developers understand it better.

For each metaphor:

* Create a list of aspects related to the metaphor
* Transfer these aspects to your target system
* Determine which of the aspects might be useful

When experimenting with prototypes, you should use the standard procedure when working with prototypes:

1. Planning
   1. Describe the prototype content
2. Development
   1. Start with simple prototypes on paper
   2. Then simple prototypes in for example PowerPoint
   3. Then functioning prototypes
3. Preparation
   1. How much should you cooperate with user during the prototyping?
   2. How realistic should the prototyping be simulated to be?
   3. Which users should be included in the prototyping?
4. Test
   1. Use the prototype
   2. Observe
   3. Document everything
5. Report results

If this procedure seems contentless, it’s described in detail in the notes from an earlier course.

**Systems**

This is where we choose which IT-system we’re going to develop.

The two proceeding sub activities aim to declare as many interpretations and possibilities as the situation allows for. What we now need to do is to concisely and accurately define these possibilities and the best way to do that is by authoring **systems definitions**.

A system definition describes a complete entity; it tries to gather and combine all different interpretations and perspectives into one concise and accurate description. It puts the focus on how everything should work together. It doesn’t concern itself with details like how should the database be organized, but concerns itself with all-including properties like “User friendliness”. User friendliness is only a relevant term for the system as a whole. No part of the system possesses this property on its own.

In the same way, a system definition also reflects certain boundaries that have been made.

See page 36 in the book for an example of a system definition.

The following are some practical tips for making a good system definition:

* Use general terms and focus on general properties
* Focus on ideas rather than the current situtation
* Make the definition concise and accurate
* Experiment with several system definitions
* Do not get caught in the mental tunnel-vision; think in new ways, criticize tradition

## The FACTOR criteria

The FACTOR criteria (or BATOFF in the danish book) are a set of criteria (see the book) that any system definition should include. You can use the FACTOR-criteria in two ways. Either use it to support yourself during the making of a system definition, or make the system definition and then check if you satisfy all criteria.

## Evaluating and Choosing a system

It’s not the developers job to choose the system. While the developer might be academically invested in choosing one system over the other and wants to argue this point, it is ultimately the users choice when choosing the system.

The choice of a system happens very early in the development process, and this can cause issues. During the process the developer may realize that changes need to be made, and this will require a re-negotiation of the system choice.

## Knowledge domains in Systems Development

We need abstract knowledge to understand a domain in the context of the situation. We also need concrete experience to understand the user’s situation and discuss future changes.

Take a look at slides 8 to 10. They give a good and concise description of the different areas of knowledge and the tools and techniques you can use for acquiring abstract and concrete knowledge.

## More on Rich Pictures

As said before, a rich picture represents how the illustrator understands the situation.

A rich picture either concerns itself with change or stability - how a situation should change, or how it currently is. It’s a great way to facilitate interactions between users and developers, and helps the developer organize their understanding and get a broad overview of the important aspects of the problem situation.

When drawing rich pictures, use the following:

**Entities**

This can be people or places. Remember that roles and tasks tie people together. More on this under Processes.

**Processes**

In a rich picture, you usually use arrows to describe information processing, work and production, planning, control, or development and organizational change.

**Structure**

In a rich picture, you usually use lines to describe a structural relationship between entities.

This can be used to describe production and application, communication and agreements, ownership, membership and power relations all between different entities in the picture.

**(Advice)**

Rich pictures should generally contain much information and be open to interpretation by whoever is regarding it. It should coherently present processes and structures and show at least one problematic area. It should point to several computerized systems.

It should illuminate key aspects of a situation in a way that promotes the audience’s understanding at many levels, and avoid representing data and data processing only (programmers tend to do this).

This is a lot of information, where the appended “rich” comes from, but it should only be rich of information, and not chaotic.