## **Construction, evolution and prototyping**

The construction approach is a top-to-bottom approach that aims towards analyzing a specific problem formulation rationally and presenting a set requirement specifications from this analysis. Then the system will be developed based on these specifications. Very straightforward.

Another approach is the evolution approach. This approach is built more around the empirical mindset, as opposed to the rational mindset, where trial and error pave the way of progress during development. Instead of making requirement specifications, you then make a prototype of your system, typically a mock-up of a user interface. You then try it out with the future users of the system, and listen to their feedback. Then, you try another prototype built with the future users’ feedback in mind. Iterate until a fitting prototype has been found, whereafter the system will be built.

The two approaches should not be regarded as alternatives to each other, but should always be sought to be combined instead. So where and when do we use the different approaches?

The construction approach is well suited to deal with non-changing, complex problems with low uncertainty involved. In practise, this could be:

*“Build a program that finds the x’th fibonacci number, based on the fibonacci algorithm.”*

The problem is (sort of) complex, in that it involves a complex mathematical algorithm that has to work exactly as intended by its mathematical theory. It is also static/unchanging, in that the algorithm and the set of numbers we should be able to calculate never changes. The environment of the problem don’t change. This also means that the uncertainty surrounding the environment of the problem are very low.

The evolution approach is well suited to deal with problems with high uncertainty and low complexity. For example:

*“Design a user interface for a program that finds the x’th fibonacci number, to be used by IT-project managers.”*

Now, the problem has a human aspect, seeing that the future users are now a part of the problem formulation. With a human aspect like this, high uncertainty comes with it. There is no all-applicable method for designing a perfect UI for any program, which is why the evolution approach is well suited for tasks like these. Also note that the problem is not very complex; there are no complicated algorithms to take care of, not a lot of objects and relationships that need to be simulated. Just build a UI.

What the evolution approach does is that it makes a prototype of the UI for the program, and through feedback and new prototype iterations, it reaches a fitting prototype that is approved by the future users needs and wants. From there, the actual implementation begins.

Naturally, this also means that it’s difficult to set a definite timeframe and deadline for a project completely developed through the evolution method. This is why the construction and evolution approach are usually run in parallel during development.

**TL;DR: Use the evolution approach for low complexity/high uncertainty problems, and the construction approach for high complexity/low uncertainty problems.**

## **Experiments with prototypes**

A prototype, as mentioned before, usually contains one or two of the main components in an OOA&D modeled system; i.e, one or two of the three user interface, function, and model components.

A prototype is usually limited in a set of different ways, be it functional, graphical or in terms of the modeled problem field.

Usually, a the experiments performed with these prototypes are split into two categories:

**Exploring experiments** uses a prototype to develop new design ideas.

Discussions may be around presentation prototypes or proper prototypes.

**Evaluating experiments** uses two or more prototypes to evaluate and choose a specific design. Prototypes proper help clarify this.

**Evolutionary prototyping** is a continuous process where the system is developed incrementally. In the end, the result is a pilot system.

No matter what experiment we’re performing, the approach should always systematically follow these steps:

1. Planning
2. Development
3. Preparation
4. Field testing
5. Conclusion

### Planning

Planning starts as soon as we realize that we need more information about the system. We need to define, in clear statements, what answers we need. For example “Can the user do X manually, or should X be automatized?”, “Can we make X work?”, or “Do the users want X?”.

During the planning phase, the focus of the prototype also has to be defined. We do this by defining the focus in terms of the three main components (UI, Function, Model) and by asking ourselves the following questions: “What are we focusing on in this experiment?”, “What are we purposely ignoring in this experiment?”, and “What prerequisites do we have to fulfill?”

### Development

The development works directly from the description of the prototype content. It’s crucial that the development process is quickly finished, as it gets harder over time to maintain the end users’ motivation to be a part of the development process.

### Preparation

The field testing step of experimenting is the most crucial part, and it also has to be prepared for properly. Preparation includes determining how the field test should be carried out (should a user or the programmer use the program? Should it just be a discussion between developer and user?). It also prepares for the amount of realism of the test. It finally determines who to include and represent during the test (selection of represented userbase).

### Field testing

This test is centered around testing the prototype in the chosen environment. You usually can’t interrupt and change a field test. It is also difficult to repeat a field test. Because of this, good preparation and documentation are absolutely crucial.

There are different ways to document the test. Getting the users to write a “diary”, for example from a set of checklists, is useful especially so when the developers themselves are the testing users. Not as useful with actual end-users as the testers.

Another way is to record the test with audio. Just remember to take notes along with the recording, as a test recording is a substantial amount of data.

### Conclusion

Here, we need to conclude the results in terms of the question we started asking. The results will either be the inception of new experiments, or a sketch of a certain design. The results can be the building stones of a formulation and choice of system definition

**TL;DR - fuck you read it**

## 

## 

## **Lecture**

**Construction**

* Bureaucratic approach to systems development
* The methods used remind us of mathematics; rational, logic
* Relies on overall plans
* User plays a passive role; they provide information, approve decisions
* It’s a linear approach, no iterations
* Works well for **stable, unchanging,** and **well defined** problems; low uncertainty, high complexity
* Does not respond efficiently to change
* Closely related to the waterfall approach

**Evolution**

* Realizes that real world problems are rarely clear and precise
* Deals well high high uncertainty/low complexity
* Not suited for tight planning as it restricts possibilities
* Trial and error is a very important process in reducing the uncertainty
* Recognizes and emphasizes uncertainties
* The result is a *satisfactory* version of the system (**not** a perfect version)
* After this, the system may be developed further
* Deals with close communication with the users
* Works very well in changing environments
* Difficult to set a deadline
* Not a linear approach, iterations are key

There are several reasons for uncertainties to arise. One is simply the entrance of a human aspect in a problem. Another is simply misunderstanding the problem. Or misunderstanding how the application is going to be used.

During each iteration, keep in mind how it will affect the modeling of the problem domain (Model), the functionality of the system (Function) and how the User Interface works (Interface).

When experimenting with construction (requirement specfication) vs. evolution (prototypes) we saw the following advantages:

Construction

* More robust code and system
* Well written application

Evolution

* Better human-computer interface
* Quicker delivery of a functioning system
* Half the code
* Les robust code, less functional
* Less effort on planning, more effort on fixing and testing
* Lack of specifications

When choosing an approach, consider what factor (robustness, ease of use, etc) is important for the system to get right.

Construction is mechanistic in its world view, evolution is romantic.

**Complexity:** You have a lot of information, complex problem, but it’s well defined.

Strategy of approach: Categorize and abstract, divide into sub-problems until each sub problem has a simple solution

This is construction

**Uncertainty:** You have insufficient or unreliable information about the problem and/or its solution. The requirements are unknown or partly described.

Strategy of approach: Collect more information about the problem and solutions, experiment with different solutions. Use information from experiments to iterate and improve until a satisfactory solution has been found.

This is evolution.

The principle of limited reduction: If you reduce the uncertainty, complexity is heightened, and vice versa.

## Prototyping principles

**Clients:** initiates, makes agreements, signs contract

**Users:** work with or use the application system

**Software manufacturer:** develops and delivers the application system

**Prototype:** an early version of the system, is operational, is a communication medium between users and developers

**Presentation prototype:**

* Developed quickly using few resources
* Early
* Determination of overall requirements
* A first impression

Developed using very few resources. A typical presentation prototype is a paper prototype. It can also be made as a powerpoint with hotspots for mouse clicks. It contains no real important functionality.

On purpose is to see how intuitive the system is. This is why it’s good to include the user in the process, but vocal support during the prototyping hinders our ability to see how intuitive it is.

**Prototype proper:**

* Provisional operational software system
* Illustrate specific functions
* Reveals design problems
* Realistic

Contains central functionality, as opposed to the presentation prototype.

**Pilot system:**

* Not the final system, but taken into use by the users
* Sometimes used in daily work
* Peak realism
* This is where we see if the system works in the application domain

## Horizontal and vertical prototyping

In horizontal prototyping main focus is on the interfaces. You usually cover all of the interfaces, *some* of the functionality, but not yet concerned with the data models. Your focus is on the top layer of the system.

In vertical prototyping, a part of the system is implemented completely. This means some of the interfaces, some of the functionality, and some of the data modeling.

## Principles

Specifying requirements is a difficult task.

A prototype is a part of the application system specification. When the user has said that a prototype is “what we want to go with”, that prototype becomes specification for what the system needs to look like/do.

Prototypes are a step on the way (increments) to the future system.

Prototypes are used exploratorily or to test solutions and ideas.

## Cooperative interaction

**Wizard of Oz prototyping** means that **you** will be the computer. The user will use the interface, and you will simulate the reactions from the computer. You might choose to be in a different room, or to be present. It depends on the circumstances. This way, you can adjust in the moment, and analyse inputs. This is efficient, as it is easier to alter your response than to rewrite the code if something the user wanted to do didn’t work the way they wanted it to.

It allows you to do rapid iterative test of design ideas. It’s easy to make modifications to the prototype. It encourages collaboration and dialogue. It engages the user and makes it interesting.

It also has problems. The “wizard” can have incorrect behaviour. It is required that the wizard has a good understanding of the prototype and the problem, purpose of the system.

If the user becomes too aware of the wizard, it can hinder feedback.

There is also the concern of user bias, or emotional stakeholding; “this is not the way we do it!”

## Exercises

Aalborg Lufthavn

**Problemområdet:**

Indebærer alle de fysiske elementer som fly, helikoptere, terminaler, gates, ruter, luftveje, tidsplan, vejr, grundkørselsfartøjer.

**Anvendelsesområdet:**

Flyvelederen, kontroltårnets personale

Flyvelederen skal have: Flyenes position i tredimensionelt rum, fart, rute, brændstof, liste over alle fly, direkte til flyene, tidsplanen, flytype, prioritering af fly.

Flyvelederen påvirker problemområdet ved fx. At ændre et flys rute, et ændre landingsbaners tilstand (åbne, lukkede)