User Centered Program Design

Børre Stenseth, 1999

Extract from: http//www.ia.hiof.no/~borres/marketmet/

User Centered Program Design

Børre Stenseth, Dec. 1999.

This is a short presentation of a method for program design. The word design must be understood in a wider sense than user interface design. The design process as defined by Websters Dictionary [3] covers a wide approach and seems appropriate:

> The approach that engineering (and some other) disciplines use to specify how to create or do something. A successful design must satisfies a (perhaps informal) functional specification (do what it was designed to do); conforms to the limitations of the target medium (it is possible to implement); meets implicit or explicit requirements on performance and resource usage (it is efficient enough).
>
> A design may also have to satisfy restrictions on the design process itself, such as its length or cost, or the tools available for doing the design.

The method covers the process from idea to program. The focus is on "tools for thinking" in the process rather than rules for designing the user interface or the technical design of the program. The tools have a form that gives priority to simplicity and communication, rather than precision and formality. That means that **participation**, from different kinds of knowledge, experience and

expertise, is given high priority.

The method has been developed and used mainly as a tool for designing computer programs for educational purposes. The method has a wider applicability. The techniques are fairly general and easily adaptable to programs of other categories. Design of computer programs in general is in my experience to a large extent a pedagogical endeavour.

The method as it is presented here is the result of a long cooperation with my "brother in arms" Ivar Minken [8], and it is the result of a large number of projects and design courses. The background of the method is described in references [1].

The article: Pedagogy and Technology [2] gives a historical perspective to the method.

Table of contents

The Philosophy

[Communication] [Pedagogy] [User centred] [Creativity]
[Generality]

Communication

All non-trivial development projects that I have been part of, or know of, has involved participants with different backgrounds. They have represented different skills and "schools" within the programming domain and they have represented different perspectives within the application area. In projects for pedagogical design, the pedagogical experience of the participants has played a major role.

The general problem has on all occasions been to find effective means to communicate ideas and proposed solutions. Language is power, and the party that controls the description usually controls the process, and the result. This does not always end up with the best solution for the user. The history of computing does indeed demonstrate that the ideas of the computer professionals, who has dominated the design process, not always match the needs and views of the users.

It has been important to find ways to express ideas and proposals in a way that can be intuitively understood and shared. This supports a broad and substantive participation, a shared responsibility for the result and it supports the creative aspects of design.

A consequence of this approach is a demand for simplicity. Lack of precision will in some cases be the cost. In my opinion this is a fair trade-off in important phases of a design process.

Pedagogy

The approach has a pedagogical background and it is in all respects empirical. It has been gradually developed, rethought and adjusted during a great number of projects with educational or

instructional computer programs as objective. This means that the tools have been discussed and tested with respect to the pedagogical views and opinions of a lot of teachers with experiences ranging from kindergartens to universities. The products and prototypes have been tested on a great number of students of all ages.

This process has made it clear that the method cannot be pedagogically neutral. The method takes a stand for an open experimenting pedagogy and some of the main tools involved reflect this. This approach is in general terms in accordance with the dominating pedagogical tradition of the Nordic countries where the method has been developed.

User centred

The approach takes a radical stand for a design process that starts and ends with the program as a communicating device. In stead of considering the "GUI" (Graphical User Interface) as something you add at a late state of the process, we start there, and adds the contents and the data models late in the process. In fact we don't bother about this at all in this article.

Creativity

The balance between creativity and formalism is critical in software design. Most design traditions does not have any explicit tools for the creative part of the design process. It seems to be a rather general opinion that creativity is an uncontrolled mental process that is substantiated either as unplanned brainstorming sessions or incidental inspiration.

My experience is that creativity is hard work and that the need for tools for thought and communication are essential during the creative parts of a design process.

Generality

Even if the empirical background is within a pedagogical setting, it is becoming increasingly clear that the approach is applicable in program design in general. The main reason for this is that pedagogical thinking is an essential part of almost all program design. The program must communicate an understandable model of data and the problem solving method it supports. The analogy between experimenting to **learn** and experimenting to **do** is evident. The designs that supports these two perspectives have much in common, and I will expect, and certainly hope, that pedagogical reasoning will find a broader platform in program design in the future.

Overview

[The Idea] [The Objective] [The Metaphor] [The Scenario]
[The Activity Table] [The Market Diagram] [The Screen]
[The Dialog] [The Details]

The method is described as a series of steps. These steps are shortly outlined below. It is obvious that these steps cannot be a one-way stride towards the final product. The process must be iterated and any experienced designer will know that the available techniques must be seen as a set of tools that should be used as needed, in a sequence that is useful for the project or problem at hand.

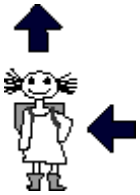
The Idea

I could have started out with "The Assignment" in stead of "The Idea". This would probably be a more accurate description of the situation in most projects. I have deliberately chosen to stress that a fruitful design process should have an underlying idea. The idea should feel good and it

should give you some assurance that this is possible, and that the result may become really good. The idea may also be the inspiration for a very important concept: the name of the project. This gives identity to your work, and it is my experience that a catchy name really helps inspiration and communication.

Thoughts and sketches that describes the idea are important design documents. Usually the idea comprises some preliminary rationale and motivation for the project.

More



The Objective

A proper formulation of the objective should answer the following questions:

- Who will use the program ?
- What should the users achieve with the program?
- Why is it a good idea to use a computer to achieve this?
- How should the use of the program be organised ?

This has two purposes. It is partly a basis for deciding if this is a project which is worth the effort, and it is partly a necessary foundation for a lot of decisions later in the project.

More

The Metaphor

Any subject may be communicated in many different ways. An important part of the creative efforts in design is to find and work through at least three different perspectives to an idea. The chosen perspective is the foundation of a metaphor.

The selection of the metaphor is probably the one most important decision in a design. It has consequences on terminology, functionality and graphical design. The metaphor should support the users cognitive model of the program, an it is the metaphor that makes a program a consistent unit. The metaphor should support motivation and communication.

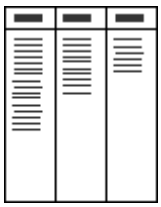
More

The Scenario

The first attempt to get a grip of the programs functionality is done by describing a scenario. We try to write down the things that will happen in a typical session with the program. This description is used to extract the verbs and to go on with the next two techniques, the Activity Table and the Market.

A scenario will never be complete and cover all possible combinations of actions, but it should pick up typical work sequences.

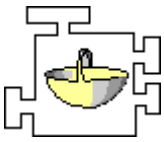
More



The Activity Table

The scenario is the foundation for distributing the responsibility for different actions between the main roles involved in using the program. The roles are dependent of the type of program. For pedagogical software the typical roles are the student, the computer and the teacher.

More



The Market Diagram

The market diagram is a graphical description of the functionality. The scenario gives us a textual, free form description, the activity table helps us to sort things out, while the market gives us a map to work with.

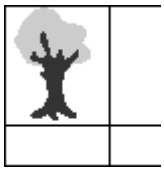
This map gives us the opportunity to discuss the connection between activities and pedagogy.

As the metaphor, market place, indicates, we do also introduce a basket in the market. We use this to symbolise the accumulative aspects of the users work, or learning.

More

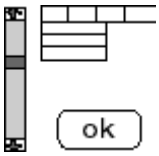
The Screen

The screen, or the program window, is the place where we will realise our idea, with the learning objectives, the metaphor and the functionality. We



must choose a presentation and an architecture that match our aspirations. It is a good idea to postpone the sketching of square window drawings until we have worked through the phases above in a free form.

More

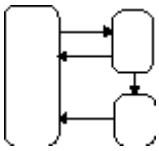


The Dialog

The program window, or windows, is the place for communication between the program and the user, the user interface. A lot of static and dynamic elements must find their form. It s a good idea to look close at the established, or recommended, standards in this phase of the work.

It is also necessary to give some consideration to the overall role of the program as a part of a dialog.

More



The Details

"The devil is in the details." Since we want to have a broad participation in the design process, we must also offer a tool for analysing the details in the program. The available tool is state diagrams.

Most details are solved by choosing standard solutions, but there will almost always be some special situations that must be resolved.

More

The Steps

The following sections will describe the steps or phases of the method. If you don't like the sequential notation inherent in "steps", you may refer to them as techniques. The description will be a combination of explanation, comments, and general experiences from their application.

The concept of prototyping is almost not mentioned as part of the methodological approach. This is a deliberate choice. I believe firmly in the use of prototyping and "program sketching" in almost all phases of the design process, and has a large personal toolbox for this purpose. The intention is to stress the fact that we need tools for thinking. It is my experience, with own work and others, that prototyping without planning often leads to programs that grows uncontrolably in many directions. Such programs very often ends up as patch work.

The Idea

[Characteristics] [Sources] [Evaluation] [Draw it]

Characteristics

Program design, as I understand it, is a more or less systematic moulding of an idea to a running program. We must put some demands on an idea that may serve as foundation for this process. Designation of a topic is not sufficient. Formulations like "I want to make a program for learning languages" or "We need a program to teach coordinate systems in mathematics" is not ideas. At best it may be characterised as an expression of want or a good intention.

An idea must of course clarify the topic that will be treated, but it must incorporate something more. We must have some thoughts about how the topic should be treated, and we must have a feel for the dynamics and the perspective.

Often a good program idea is characterised by a new and original perspective. We want to experience an intuitive feeling that the idea is different, in some respect, from known programs and earlier solutions.

It should be useful an engaging. We will of course have to test the usefulness against the objective. We should also incorporate in the idea a very coarse understanding of how the technology can contribute to a better way to work or learn.

Sources

It seems to be a general comprehension that the good ideas almost by chance drop into the heads of a few talented and creative individuals, and the only way one can hope to "get an idea" is to wait patiently. Often ideas has roots in the experience of a teacher, but good ideas may have unexpected sources. Even if the project is initiated as a result of a requirement analysis, the idea should be sought and worked on.

It no doubt happens that ideas appear apparently out of the blue and occupies your mind without forewarning and mercy. But usually generating ideas is hard work which involves both mental preparation and a methodical approach.

If there is anything like a talent for program design, this talent is a balanced combination of madness and discipline. The madness should open up for chains of thought that is not necessarily feasible or apparently fruitful. Discipline means that the designer can take two steps back and evaluate and modify the thoughts in a systematic manner. In a design team it is crucial to allow and nourish both these approaches.

The work with an idea will give birth to a lot of sketches that will not be used. Don't throw them away, they may be useful in later projects and the subconsciousness will continue to work with them as sources for inspiration.

A characteristic of professional design is that the ability to be creative can be maintained even when that first elation has passed and things get difficult and boring. This always happens, and it is important to be ready for it.

Evaluation

We must analyse different aspects of the idea. One important issues is that of technological limitations. It is a clear experience that most people tends to underestimate the technological possibilities when it comes to concrete program design. This goes for computer experts as well as other members of a design team. We should try to avoid this kind of technologically based self censorship in the start of a design process.

The main thing to look for is the simple aesthetics and consistency. This is to a great amount a question of intuition.

Draw it

To pursue an idea means two things. First you must make it clear to yourself. Secondly you must make it clear to others. For both purposes the idea should be materialised in one form or another.

The description should be short and to the point. It is very easy to get into discussions with yourself or others about details that is better resolved in a later stage in the work. As a rule it should be possible to communicate the idea on one blackboard or one big sheet of paper. The description should consist of little text and many sketches. If the idea is hard to draw, it should be considered a warning signal. Diffuse ideas are hard to draw. Avoid square drawings with window frames. This will lock up your design at a too early stage.

Baptise your idea. It is much easier to work with an idea with a catchy name. Tell it to others. This will most certainly make it easier to communicate about the idea, and you will get questions and suggestions from your colleagues.

Sample

The Objective

[Who] [What] [Why] [How] [Maintenance]

The objective of the program should be formulated as early in the work as possible. A reasonably precise objective forces us to make important decisions and it supports these decisions. Such an objective should go beyond that of a requirement analysis.

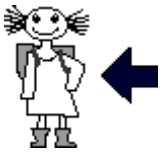
The objective must be formulated before our project can be evaluated relative to usefulness or in terms of pedagogical and commercial criteria.

The objective will fill many purposes:

- If it is sufficiently precise it will be an important part of the foundation for all decisions

regarding both content and form.
- Questions about limitations will appear throughout the whole design process. The objective is a good tool for decisions of what should be in and what should be thrown out.
- The objective is the foundation for the metaphor we choose. The concepts and the images that we use must correspond to the user profile we establish as part of the objective.
- The objective is also an instrument for evaluation of the product. Operational objectives supports a precise evaluation.
- The objective is the raw material for marketing of the program where who and what is the basic questions.

A systematic approach is to formulate the objective as answers to the following questions: **Who**, **What**, **Why** and **How**:



Who

Who are the users? The same way an author must have a clear picture of the reader, the program designer must anticipate the skills and interests of the end users.

In educational software the solution may be to formulate this as the users age and grade. This may however be too indirect for our operative needs. It may be better to describe the necessary prerequisites of knowledge or abilities. In programs with other purposes the description of the users may find other forms: Language, age, experience, cultural background, education, occupation etc.

What

What should the user achieve by using the program? This is in many ways the central part of the objective, and in most cases the most difficult part to formulate. It is however worth while to try to get it as clear and operative as possible.

Pedagogical literature contains a lot of material for formulating teaching and learning objectives. The planning of a program differs from general pedagogical planning in that the demands for precision is probably greater. If you consider a program as "canned" pedagogy, the possibilities to be adaptive in your conduct is dramatically reduced.

In some cases it is possible to formulate objectives that may be measured with a reasonable degree of precision. For typical tool programs it may be possible to measure efficiency in terms of time on task.

Insight
Knowledge
Skills
Practice
Information

It may be useful to consult the taxonomy that is established for cognitive and practical achievements. Se for instance Bloom [5]. The formulation: "The user will get insight in .." is much more of a challenge than the formulation: "The program will give an overview..".

Overview

Why

We should take the time to formulate a reason why we want to use a computer for the task at hand. In many cases this is implicit i all the reasoning we do, but still it may be worth while to test our thoughts by writing them down. Maybe this will reveal that other teaching or learning methods or materials are better ?

We must investigate if we are reinventing the wheel. We should also do a survey of existing products in the same category, or products with similar what-formulations. The task may have been solved by a program already, or our objective may be achieved by combining existing programs.

How

Early in the work we should give some thoughts to how the program will be used. This goes for all programs, games, tools, pedagogical programs etc. Some relevant questions:

- Is the user alone or will others be present? Others may be teacher, client, supervisor, student, customer, depending on the type of program.
- Do we have another person as a necessary participant ?
- Will there be any sources for stress when using the program? Customer on the phone or at the counter, time limit on task, etc..
- Will the program be used as an electronic blackboard for demonstrations or instructions for groups ?
- Will the program depend on a pedagogical plan that must be staged for the program to be effective?
- Should the program be sold at the gas station and be completely self contained?.
- Will we be based on updates of data from the internet ?

Maintenance

The objective must be maintained during the design process. It is not necessarily a defeat if we decide to adjust the objective as we dive deeper into the project. It is a common experience that the work will reveal possibilities for broadening the objective, both with respect to who, what and how.

Sample

The Metaphor

[Change Perspective] [Well known metaphors] [The naive physics]
[Simplification]

In this phase the design may take unexpected directions. It is a good investment to put a considerable amount of effort into the choice of metaphor. The metaphor is the setting where we want to place the user. The metaphor is in many ways the personality of the program. A good metaphor is essential to both the user interface in traditional sense, as well as to the work to be done and the learning to take place.

The concept metaphor is used widely, in different contexts. Many are familiar with the biblical metaphors, which are used to communicate complex moral and ethical issues. Poets use metaphors, filmmakers use them and they are a part our daily language.

Within the field of program design the use of metaphors is established as an important issue for many reasons.

Communication

Some designers focus on the metaphor as a rather shallow illustration that mainly serves to explain the functionality of buttons, messages and menus. The design of the metaphor is in this understanding mainly a question of designing buttons and texts.

Navigation

Navigation may mean at least two things: navigation in functionality (*where is the button that turns of automatic spell checking?*) and navigation in data structures (*where is the comments I wrote yesterday ?*). The use of the Internet has set focus on the latter, and has raised a lot of interesting questions regarding consistency in metaphors.

Understanding

A good metaphor contributes beyond communication and navigation. It should support a cognitive model of the program. We should know what to expect and we should be able to foresee the consequences of a series of actions. It is important to design the metaphor such that it rewards understanding. The users efficiency increases as you gain experience with the program.

Learning

In a pedagogical perspective we do also want the metaphor to support learning. The metaphor should support the cognitive process that involves learning, and it should be adapted to the cognitive level we have stated in the objective. A phrase like "The user should understand..." is much more demanding than "The user should get an overview...". This difference must be reflected in the metaphor.

Motivation

You may have different approaches to motivation. In a general sense the program should sell itself. The argument can be rational (*it is effective and saves time*) or it could be irrational (*it looks nice*). The first takes some explanation, or experience, while the latter is based on first impression. In a pedagogical perspective there is a parallel reasoning. On one hand the program can be designed to be part of a pedagogical plan and work effectively within that plan. On the other hand it may be designed to be self motivating and function without planning or a special context. The latter approach is of course the difficult one.

Place
Role
Time

Change Perspective

When we start out with an idea, preferably a drawable idea, we do usually have a rough metaphor in mind. It may be incomplete, it may be unconscious and it may turn out to be a dead end for a lot of practical and conceptual reasons. It is important to investigate other approaches, other metaphors.

It is good rule of thumb to come up with at least three different perspectives. Changing perspective is generally a rather difficult mental process and we need some tools for thinking that can support this process. One way to do it is to work systematically with **place**, **time** and **role**. Make up a perspective from descriptions of these three key concepts.

Place

When we read a novel we are mentally carried away to another place. The author wants to do this to us to prepare us mentally for the plot or the narrative. We must consider where we want to place the users of our program.

The options are usually plentiful if we take the trouble to look for them. In a program about cultivating of fish we may choose to be at the pond (seeing the fish) or we may be in the managers office (seeing the balance sheets and the plans). These approaches gives two quite different programs, even if the objective may be the same.

All the programs that I know of have all made decisions about place, deliberate or not. Word processors take their metaphors from "in front of a typewriter", while layout programs puts us in front of the "light board" where we can paste up pieces of text or images. Even if the functionality is getting more equal for each new version of the two program types, the basic concept, the place, is different. An intriguing issue is that both the typewriter and the light board is getting less and less known for most of the potential users; the frame of reference disappears.

Role

The same way that we can look for different places, we can look for different roles. We can look for roles independent of the places we discussed above. In the end the role must of course match the place, but we are know looking for effective approaches to stimulate our creative efforts for alternative perspectives, and finally a good metaphor.

In a program dealing with traffic rules we may be the driver of a car, a police, a pedestrian, or a student in the teachers classroom. These perspectives will probably lead to fundamentally different programs.

We may also analyse the role in a more direct way. We may look into different role models for the user towards the program. This is a concept that are often overseen. We may as well put the student in the role of a teacher as in the role of a student.

Time

Time is an important and difficult issue in many aspects of program design. We may discuss time as part of the user interface design and we may discuss time as part of the metaphor.

The use of time in a model may be rather complex and difficult to grasp. We may think of an ecological simulation as a case. Are we moving faster than real time, slower or do we make leaps in time. How should the user affect the time ? Can we use retrospective techniques, playback or simulate ahead to illustrate consequences of decisions ? Usually the problem is not time as inherent part of the model, but the users understanding of time and orientation in time.

A rather straight definition of time may be in place: Frank, a character in the film "Late for dinner", after having been frozen for 30 years: "*Time is something we have to prevent all things from happening simultaneously*".

Well known metaphors

Often the search for a metaphor may seem like an unnecessary effort or some kind of conceptual overkill. You will hear designers claim that " this stuff is self explaining and the program will do fine without a metaphor" or " this is a serious program and a metaphor will just make it look childish".

It is important to note that all programs is based on a metaphor of some kind. The choice may be deliberate or not.

A drill or test program that invites the user to answer by choosing from multiple choices, is based in a teacher - student metaphor with all the consequences that may have. The designer may not have considered alternatives.

The programs that we know as successful programs all have a deliberate metaphor that is instrumental for their success. It is interesting to look at a few typical samples that we all know in one form or another.

The Spreadsheet

Very few programs have had an immediate success like Visicalc, the first (?) commercial spreadsheet program. The concept of evaluating a set of equations is as old as the computer, but the idea of placing variables, constants and functions in a table made the application understandable and usable. The concept of a table for organising made the difference, and we all have a conceptual model of spread-sheet programs as a table.

The Paint program

MacPaint from Apple introduced an interesting metaphor, which we may call the painters metaphor. The idea of the canvas and the available tools was almost intuitively understandable. Two tools was a little hard for new users. The hand that should position the canvas within our view port, and the lasso. The latter is a intriguing element i the metaphor, and most users had problems grasping it. The fact that it is a reflection of a useful concept (regions) in the graphical toolbox did not help most users. This anomaly was accepted because the rest of the metaphor was consistent and communicating.

The Desktop

The desktop is interesting in many ways. The most interesting thing is the gradual transition from an attempt to mimic a physical desktop towards a concept which defines its own attributes. The desktop has i many ways taken the route that many metaphors in the language does; from an explicit analogy to a part of the language. When the Internet gets incorporated in the desktop in a seamless way this transition will be even more clear.

The naive physics

The metaphor has, and should have, what we can call a naive physics. It should be evident what we can do and what we cannot do. The metaphor should match our earlier experiences and we should be able to act without any further explanation. When we see a dice on the screen it is obvious that we should be able to roll it. If we see a card it is obvious that we should be able to turn it or move it. If we can not, we are puzzled and we loose faith in the metaphor, the model and the program.

Simplification

Design is an iterative process and it should contain phases of expansion and phases of reduction. When we deliberately stress the tools for

supporting creativity without limits, we must accept the cost of having a set of rather colourful ideas, metaphors and thoughts. The good thing is that we have something to choose from.

The metaphor we choose will be subject to modifications, and most of all reductions. Typically we will start out with a metaphor that is rather colourful and detailed and which has associated to it a lot of images in the back of our head and hopefully on paper. Only part of this is ultimately suited for presentation on a screen, and we must be prepared to reduce the metaphor down to the essentials in later stages of the design. Someone has once said that a design is not perfect when there is nothing more to add, but when there is nothing more to take away.

Sample

The Scenario

[The Verbs in the story] [The Nouns in the story]

At some early stage in the design process we will have to start thinking about the functionality of the program. Normally we have a coarse idea of this already, when we work with the idea. It is however necessary to focus on this and attempt to make a fairly complete list of the functionality we want to offer the user.

This methodical approach offers three different tools for working with functionality:

- The Scenario (text)
- The Activity table (table)
- The Market Diagram (graphical)

We will start with the scenario. What we want to do is to try to write down what is going on in one or two or three imaginary sessions with the program. This technique is well known, independent of the design approach we apply.

The important, and difficult, aspect of this is to choose the right level of detail. If we go too much into detail, we tend to lock the functions up to a concrete interface design. If we don't include any details we don't get any effect from the description. The key is to be fairly detailed, but to keep the language in line with the metaphor. If we have chosen a book metaphor in some program we can write: " the user turns a page and inspects the suggested solution". If we write: " the user clicks on the next page button to see the suggested solution", we have decided that the turning of pages is implemented as buttons. We should try to avoid jumping to this kind of conclusions at this stage in the design.

Let us look at a very simple example: A part of a scenario for a program that is meant to be an editor for putting together chess articles in a newspaper:

> "...the user then makes a new move on the chessboard and the program update the list of moves. The user places an image of the board in the test column. He then writes the final comments. Before closing the session he decides how many columns he want in the article and inspects the laid out chess article for the week. Finally he saves the article."

The Verbs in the story

The verb phrases are:

make a move
update (list)
place (an image)
write (a text)
decide (the work)
inspect (the work)
save (the work)

These are the things we take with us for analysis. We want to identify synonyms and we want to identify the roles responsible for the different actions. This is the foundation for the activity table and the market diagram.

The Nouns in the story

It is obvious that we have also given names to some important objects in the program: (chess) board, move, article, layout, column. This is, hopefully, concepts that are part of the chosen metaphor. We can use the scenario to check out whether the metaphor does cover the concepts we need.

An other interesting path to follow is to use the nouns as starting point for an object oriented design. We will not follow this path in this text, but it may take us into very interesting terrain. We may find ground for thinking pedagogy in quite another setting than we do in the market diagram in a later section.

Sample

The Activity Table

[Student] [Program] [Teacher]

The list of verbs from the scenarios is organised and tabulated. Identical operations are identified as the list is compiled. We want to focus on all the participants which are described in the scenario. The number of participants, and their roles, will vary from program to program. In a typical pedagogical program the participants will be either, student, teacher and program or simply student and program. Each of these roles are responsible for some actions.

The distribution of responsibilities reflects the roles we have given to the user as we discussed under the metaphor phase.

Student	Program	Teacher
selects a colour moves a card writes a comment	shows the progress updates the table	sets up an assignment

Student

This is usually the leading role. We expect most of the actions to be found in this column.

It is important to find clear and unambiguous formulations. The realisation of our pedagogical ideas will be reflected in this list. It should ring a bell if the list mainly consists of entries which starts with "Select...". This tells us that the program will not invite to much activities

and initiative from the student.

Program

This list of actions or responsibilities does not add much to the program description in a technical sense. The important thing is to clarify what the program should do, as opposed to the other roles.

If we say that the program shall "Show the words to choose from", we have clarified that it is not the users responsibilities to remember or figure out this words. That is an important decision in the design.

Teacher

The length of the list of the teachers responsibilities will vary from one program to an other. In some programs we will not make the program dependent of the teacher at all. In other cases the teachers role may be advantageous but not necessary. A typical teachers role is to prepare material or to be available for consultations in the knowledge domain that the program is designed and used for.

It is also a general and rather interesting experience that responsibilities that originally is given to the teacher, during the design process ends up in the students column.

There is a potential danger that the teachers role is described as that of a user manual. An incomplete or bad design can not be rescued by a living users manual.

In general it is difficult and hazardous to anticipate a very integrated teachers role. The program will be used under circumstances that the designer cannot foresee. This circumstances may not be optimal for the purpose, but the program should do the best of it.

Sample

The Market

[Syntax] [The Basket] [Pedagogical implications]
[Types and pitfalls]

Running a program is like shopping in a market. We can shop in an open market or we can get

entangled in a bazaar without any understandable structure with narrow passages. We will use this market metaphor to describe and discuss the functionality and the pedagogy of a program.

The market diagram is one of the most important documents in the design process. It helps us:

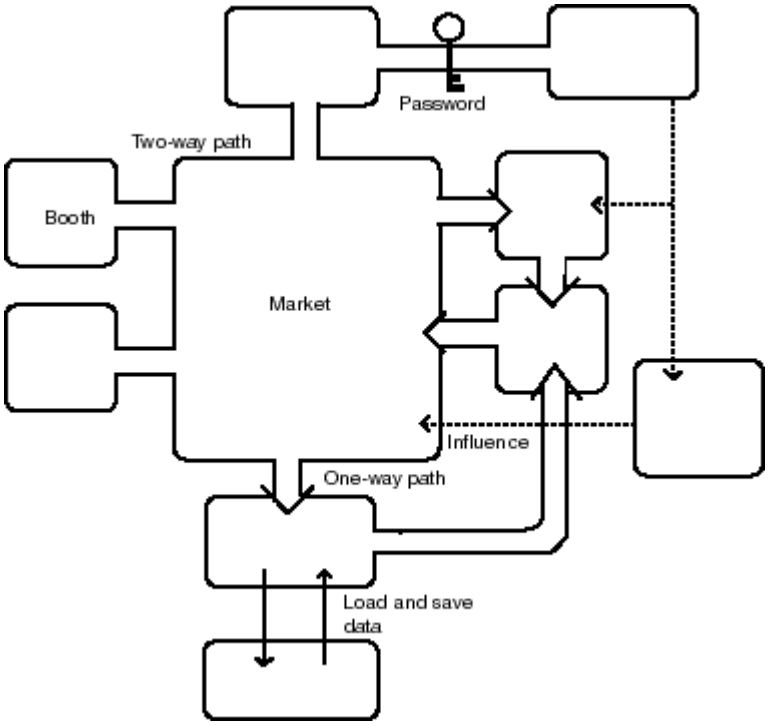
- analyse functionality
- discuss pedagogy
- communicate within the project

The market diagram is a compromise between precision an overview. Trained programmers and system analysts will find that the diagram lacks many elements we would expect to find if it should be a precise tool for program descriptions. The most important is that it has no syntax for describing conditions, and that data flow is only described in a rudimentary way.

When the diagram, in spit of this severe lacks, play an important role in design is it because it supports an holistic approach, and that it enhance communication.

Syntax

The main elements in a market diagram is one or more marketplaces with shopping booths. The availability and the connections between these elements describes the functionality. It describes how our shopping can be performed. Connections from one area to another may be open, two way connections or they may lead us into paths with no return. Some connections may need a password. To perform an action in the program is analogue to entering a booth in the market.



The limitations are obvious; we have no conditional paths, we have only rudimentary tools for indicating data and models. What we do have is a visual metaphor that enables us to plan and communicate the overall structure of our design. And that is basically what we are looking for.

A booth is **not** a storage room for data, it is a place for action. The

actions from the activity table finds their places in booths in the market.

Markets have different architecture. We will find open markets and sequential markets. The intention with the market diagram is to be able to identify the architecture and discuss it qualities in respect to our functional and pedagogical objectives.

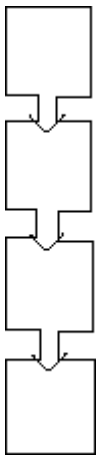
The Basket

If we stay in the metaphor, it is clear that we need a basket to carry with us the things we shop in the booths. The market metaphor thus lets us discuss the learning that takes place as accumulating experiences from the use of the program.

This may seem like a natural and uncomplicated train of thoughts, but the concept of the basket is a rather difficult and challenging one. We may have at least two perspectives on the basket:

- We may consider it as a natural part of the metaphor we have chosen for the program, that helps the user orientate herself in the structure of the program and/or in the accomplishment of the task at hand. This is a design challenge, but in many cases this boils down to simple techniques that are well known. (percentage completion, maps, indicators etc.).
- On the other hand we may associate the basket with a cognitive model of the users accumulated learning. It is a far more challenging task to find elements in the program, preferably from the chosen metaphor, that supports this interpretation of the basket.

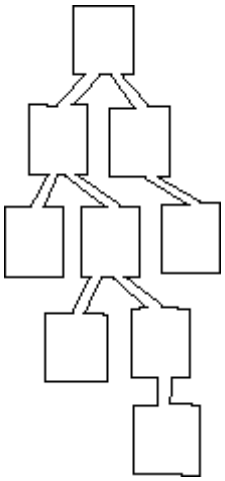
An interpretation of the basket as a symbol for the students portfolio may be fruitful and we can find some interesting theoretical approaches that may help us grasp this concept. This discussion is beyond this text.



Pedagogical implications

The structure of the market discloses the pedagogical reasoning in the design.

A market that contains long sequential rows of booths reflects the designers urge to control the learning. The actions must follow a planned sequence. This is a typical problem in a large number of designs. The designer has in the back of her head an idea of what is the best sequence to do things, and usually the argument is that "it is not possible to understand this before the student has done that". This is in many designs the core of the matter, and where the pedagogical challenge of the design becomes evident. In almost all designs it is possible to break these sequences and find another approach. An argument for an open experimenting pedagogy is an argument for an open market.



The markets with large hierarchical parts usually disclose a design where the structuring of data or functionality has been the major priority in design. It is not necessarily a connection between these structures and a learning strategy.

Both the sequential and the hierarchical approaches find their applications in areas where drill and training is essential. There may be good reasons to consider very controlled learning of critical work sequences in for instance industrial security applications.

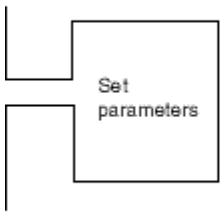
In general the method is based on a philosophy that promotes the open, free market. We want the user to control what she wants to do, when she wants to do it and in which sequence she prefers to work. The user should be in control.

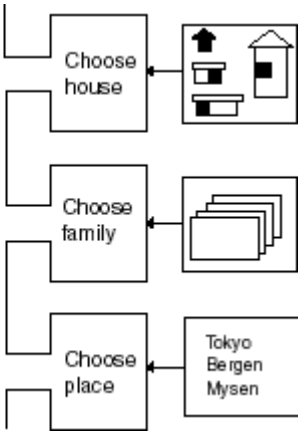
Types and pitfalls

Experience shows that there are a few things to look out for when working with the market diagram.

Details

All the techniques should contribute to the progress of the work. To achieve this it is important to find a reasonable level of detail. It is very easy, and in a sense correct to draw a market diagram like the one on the left.

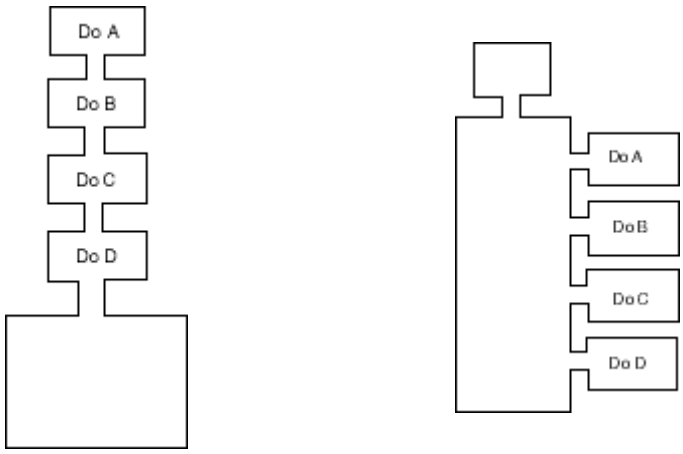




The one on the right does however opens up for a more detailed discussion about the activities.

Swan necks

A very typical discussion in design of "canned" pedagogy is that of sequence and inherent dependency in the material. Some things does not have meaning until something else is understood, or done. This kind of reasoning often ends up in the typical swan necked diagrams, as the left one below..



It turns out that it is almost always possible to find a solution as on the right figure. It has rather obvious parallels to general program design; it would be rather awesome if a modern word processor should prompt us on all possible settings before we would be able to start writing.

Standards

Most programs has some standard situations that should be treated in a standard way. If we have some kind of document based application, it is reasonable to expect that we will have the usual, new, open, save functionality, and there is no reason to overload the market diagram with these details; a simple archive booth would do.

Sample

The Screen

[The foundation] [Stability] [Surroundings] [Main Architecture]
[The Basket]

The screen or the program window is the place where it all should come together. We probably have a lot of documents, sketches and text, and the challenge is express all our ideas in an area a few inches across. We will find endless amounts of literature on screen design and user interface design. This literature describes techniques, aesthetics, psychological issues, readability, techniques for attracting attention and a lot of other useful issues. Some of the sources that I find useful are referenced [7] at the end of this article.The purpose of this section is not to summarise or extract essentials from this literature. We will emphasise some elements that is connected to the other techniques presented in this article.

The foundation

Let us look at some of the raw material we should have by now.

The metaphor will provide the most important material for the screen lay out. We have sketches in different forms and we have a vocabulary. We have identified places and roles. The metaphor will give us the general setting and it will give us the words to use in menus and screen texts. The challenge is more than anything else that of reduction. There is usually no point in making illustrations look as natural as possible. We will usually be better off with simple illustrations which

focus the important issues. There is a general rule, in literature as well as in program design, that all elements should have a purpose. We discussed earlier the simplification of the metaphor, and this will continue when we want to realise it on the screen. Maybe our colourful drawings will be reduced to some simple lines or even a few text elements.

The market and the activity table will provide the functionality, and to some extent the structure of the functionality. We will be able to identify actions that should be implemented as direct manipulation and we will have a good basis for grouping menus. We will most certainly find that many actions may be doubled, that is we can let the user achieve the same result by different means, for instance a menu and a button and a short-cut key.

The objective should be reflected in the metaphor and the functionality, but we may want to check the decisions we make during screen design directly against the objective.

Stability

The idea of an open market that gives the user great amount of control and freedom in her work must be reflected on the screen design. The design should be open, it should support overview and it should be stable. This means that the layout should not throw the user into sudden, unknown surroundings. All actions that requires special information or detailed functionality should be short trips that brings us directly back to stable, well known surroundings. Usually such special "booths" are realised with modal windows that temporary covers part of the stable, lasting program window.

Surroundings

If we look back in time the natural, and a few decades ago the only, strategy would be to let our program occupy the entire screen. We could then rely on having the users full attention and we could have full control of all details like colours, text etc. Even if we still can do this, it is a strategy that we will apply only in very special programs. The situation has changed in two steps.

The desktop metaphor and the possibilities to run multiple programs simultaneously force us to adapt our design to the surroundings. This emphasise the use of standards wherever possible. Our program must behave like other programs. The argument for stability must be extended beyond the interior of our program.

The Internet has contributed even more to this dependency of the surrounding. Our program may execute as part of a web-page or it may execute parallel to web pages. The idea of grasping the web as a "big computer" adds an extra dimension to our thinking about the visible interface of our program.

The principle of stability is as valid as ever, but the need for adaptability is a an additional requirement. This requirement will be even more important as the concept of a computer will change, and our program should maybe perform, in one way or another, on mobile telephones as well.

Main Architecture

The principle of user control and the urge for stability indicates that some time should be spent on planning the overall architecture of the program window. We should try to identify areas in the window and try to group functions that conceptually belongs together. This is again a reflection of the or structure of the market. The sample at the end of this text may serve as an example of this. The screen is clearly divided in areas that is easily identified, and in turn supports the dialog. The architecture of the screen matches the layout in the market diagram.

The Basket

We introduced that basket as an important, and difficult, part of the market. We tried to use the basket as a metaphor for the accumulation of the learning that take place or the work that is done. This concept should find its place in the screen design, either as an explicit symbolic representation or as an inherent part of the design.

Sample

The Dialog

[Characteristics] [Dialog types] [The Psychology of the Dialog]

We have incorporated a section on dialog, in addition to the screen design. Usually the two themes, *Screen Design* and *Dialog*, is covered by the concept *User Interface* in the literature. The reference section at the end of the article give some references [7] to sources that I find useful.

The reason for discussing the dialog as such is grounded in the approach which has been chosen in this article. We want to emphasise an holistic approach and we are interested in analysing the program as a whole consistent unit. This means that we are not satisfied with getting all the details in order in a "correct" way. Correct in this context is understood as based on psychological rules of thumb or standards. We want to try to grasp the holistic view of the program, even if we find both standards and rules of thumbs very useful. This holistic perspective is often lost in the work with getting all the details right. Much of the literature takes too lightly on this and refers to the need for consistency in the user interface.

Characteristics

Let us look at some of the basic premises for the dialog. The two parties, the program and the user, has quite different proficiencies for the conversation. We all know this is so, but we tend to forget it in design.

The program is deterministic. Once a situation and a stimulus is given, the reaction is predictable. The program is a state machine and the number of states are in principle predefined. The fact that the number of states may be vast and that the user in most cases cannot grasp them is another matter. The user is not, at least for practical implications, predictable. It is a common fault in design to oversee this important difference.

The designer predicts rational behaviour from the user, and neglect that the program should react meaningful on any input at any time. Programs based on dynamic simulation models are good examples: Usually the designers expectations to the user is balance a more or less complicated model. What the user does is, however, normally to test the performance of the program in extreme situations where the program normally reacts inadequate.

The capacities are different. There is significant differences in the two parties capacities in the communication. The program has a far grater repertoire for communication. It may use text, images, sounds, animation and all possible combinations. The user can in principle do nothing but type characters and point at one of a limited number of points on the screen.

The user has no use of her wide variety of communication means like

gesture, body language, fluent language, etc in this particular type of conversation. Even if we may foresee a situation that this means of conversation could be registered by the program, we will still have a problem to interpret them as meaningful "state changing input" to the program.

The consequences of this is that the program should carry a large burden of the communication, by suggesting alternative user actions which are relevant at that particular point in the dialog.

The program knows the program. The user does not. The best we can hope for is that we are able to communicate a useful conceptual model that facilitates the users part of the dialog. The program knows the limitations and the possibilities and should carry the responsibility to present them.

The paradox. If we acknowledging these three differences we brings our self into a quite a paradox. We acknowledge that the program has the upper hand in this limited dialog situation, at the same time as we want the user to be in control.

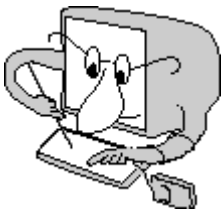
Dialog types

If we consult the literature on User Interface design we find principles like:

User in Control
Directness
Consistency
Forgiveness
Feedback
Aesthetics
Simplicity

If we see rephrase these principles in a human-human dialog context, we may say that we want to talk to someone who lets us set the agenda, is speaking a straight and simple language, think it is nice to be touched, is trustworthy, forgiving and patient, answers all our questions and is pretty. Someone to spend an evening with.

Most of us have experienced many types of dialogs with computer programs. Some prototypes:



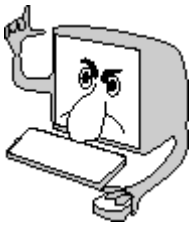
The Pedagogue

These programs are very helpful. They explain everything and usually suggests what you should do before you have had the chance to find out that you have a problem. They may even grab the mouse marker to show you how and where. There are usually one of two types of reasoning behind such programs: a detailed pedagogical plan or a desire to make a "waterproof" help system.

The pedagogical reason is probably misunderstood, and it does indeed take away the users feel of control. The user help is neglecting the fact that no one is a first time user more than once. The use of standard solutions in the interface should help us utilise experience from other

programs, and each program should take advantage of this.

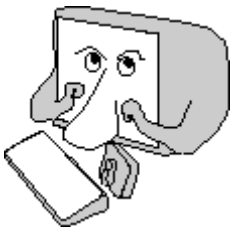
It is very annoying to be underestimated.



The Repulsive

These programs may be really though. They do not have any patience and they don't hesitate to smack your hand. Messages like: " You are not authorised to do this !" pops up without further explanations. The alternatives, to mask the possibilities out or explain the situation is not part of these programs personality.

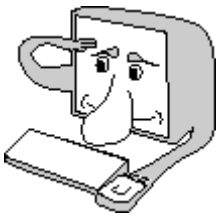
No one likes to be bullied.



The Laid Back

They really don't care. Consistence and stability is not part of their vocabulary. The content of the window and the menus changes, the same menu text may mean different things in different contexts. The modes are very diffuse and you get no feedback. Sometimes you are warned in critical situations, sometimes not.

No one like to talk to someone who does not listen.



The Helpful

The program is reacting in a polite and meaningful way in all situations. It is consistent, stable and the metaphor is precise and intelligible. It follows the standards and we know what to expect. Help is available when we want it.

It is nice to have Jeeves around.

The Psychology of the Dialog

The psychology of role assignment in computer dialogs is often neglected in design. There are some interesting research that should make us focus more on this. The book "The Media Equation" [5], reports some experiments where users react with what we could call human-to-human reactions when using a program.

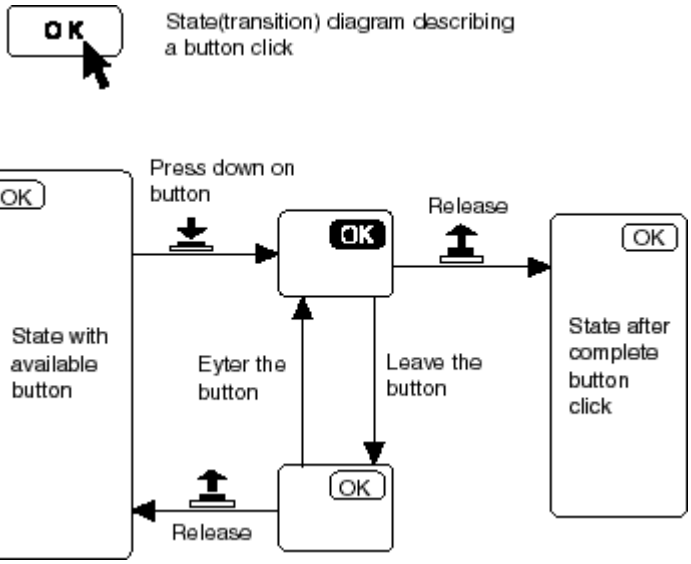
The classical example is Weizenbaums program Eliza [6]. Eliza is a dialog program that can converse forever about almost anything with very primitive means. The effect of Eliza is also rather astonishing when we consider the limited scope and variation of the conversation. Observations of users show that the users way of expression very fast adapted to the programs, and after a few exchanges of sentences it is almost impossible to tell the difference.

Sample

The Details

We have stressed that the design process should be open to participants with different background, and that the main objective with the tools we have introduced is that of communication. As a consequence of this it is necessary to offer a tool for sorting out the small details. A tool that has proven rather easy to understand and use is the state diagram, or state transition diagram.

The easiest way to explain it is to look at a very simple example:



A state is characterised as a stable situation in he dialog. The important thing is to get a map of all the possible actions we can take to change the state. One way to explain a state is to say that it is characterised by the possibilities we have to leave it.

The example above is included for simplicity. We would of course not use time and effort to analyse a standard situation like this. A slightly more complicated state transition is analysed in the sample at the end of the article.

It is important to note that we have focus on the dialog situation. We do not bother about "internal" states of the program. A lot of internal variables and data models may change without affecting the dialog.

State analyses is not an alternative to prototyping, it is a supplement. The idea is that that it should be possible to discuss alternatives without prototyping. The state diagram is available to all parties in a design group and is a very precise way to express a view on how the program should function.

Normally we would not draw a state diagram that covers the whole program, neither will we analyse all parts of a program with different state diagrams. Normally 90 percent of the interaction is predefined as part of standards without detailed analyses. There is however an interesting side to state diagrams that may be developed further. We may consider a state diagram as a map of the program where the user leaves a path when she uses the program. We could use this for testing, either by observation or automatically by logging the path. This could supply us with very useful information of how the user prefers to use the program, and how long tracks she takes to

solve a problem or perform a task.

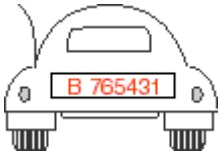
Sample

A simple sample

[The Idea] [The Objective] [Perspectives] [The Metaphor]
[The Scenario] [The Activity Table] [The Market]
[The Screen] [The Dialog] [The Details] [The Program]

A very simple program may serve as an illustration of the design process. The program "Sant" is a simple program for training aritmetic skills. Sant means "True" in Norwegian. A demo version of the program can be downloaded.

The key design documents and most of the considerations during design process are documented below.



The Idea

The idea behind Sant is Norwegian license plates as they were some years ago, with up to 6 digits. A pastime during long car trips and waiting for the bus was to figure out an equation that use all the digits. Surprisingly many 6-digit number may be "solved" this way.

7 - 6 + 5 = 4 + 3 - 1

The Objective

It is easy to see that this exercise is a training in the 4 basic arithmetic skills, and that it lays the ground for understanding equations. The idea gives us the possibility to vary the level of difficulty; we can introduce parentheses and we can vary the number of digits.

Who

After some iterations the users was described like this:

- The program should offer a challenge and a tool for training for ages above 9.
- When used in the school the program should be a tool both for those who need some extra training and for those who need a challenge.
- When used as a pastime occupancy it should offer a flexible challenge both to children and adults.

What

The final formulation ended up as:

The program should :

- train the arithmetic skills.
- enhance the understanding of the equality operators and lay the ground for working with equations.
- train strategies for problem solving.
- demonstrate the significance of priority rules and parentheses (when used at a high level of difficulty).
- offer some excitement and challenge.

Why

A computer program is well fitted for this task for many reasons:

- It is easy to produce a practically unlimited number of tasks.
- It is easy to evaluate the answers.
- It is possible and fairly easy to put the problem solving in a motivating metaphor.
- The level of difficulty may easily be varied.

How

The program is mainly designed for individual use, and will not be dependent of a special pedagogical plan or external motivation. This does not eliminate the possibility to plan the use in a pedagogical context.

Perspectives

The discussion about perspectives were mainly focused on the use of time. Two of the perspectives that were discussed in some detail are described below:

Perspective 1

Place. The original idea was connected to cars licence plates. This perspective may easily be developed to a traffic situation with cars passing.

Role. The role as passenger is possible, or we may come up with something more exciting. Maybe a police officer with the assignment to check the drivers identities or to check if the cars are stolen.

Time can easily be integrated in this perspective. The cars may be visible for a short time before they disappear around the corner or into the darkness.

This leads us into a traffic metaphor, which probably could be worked out

into a design.

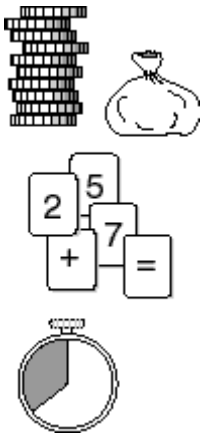
Perspective 2

Place. An other device with useful numbers are telephones. We could develop the idea of a room or situation where phone numbers had to be solved as a mean of dialling.

Role. Who would be dialling and for what reasons ?. There are lots of possible answers to this. We could take the dramatic turn and introduce some kind of spy or some role who should collect information.

Time could be integrated as a stress factor: "you must get this information within a certain time or else..."

This telephone metaphor could not doubt be developed in a more or less exciting way.



The Metaphor

There are a lot of considerations to take into account when the metaphor is chosen. One of the most important is the objective. The two alternatives above may both have interesting and in some respect promising sides, but they may narrow down the user group. We should look for an approach which is less dramatic.

The metaphor that was chosen in the end was a betting metaphor, with cards and a flexible time limit for problem solving. The user may bet with the program that she will be able to solve a problem within in a time limit defined by the user. The shorter the time, the larger the odds.

The problem is presented as cards with digits. The cards are face down and the user can set the time and bet an amount of coins. (It turned out that betting coins was not a good idea in all countries, so the betting was done with marbles in the final version, but we continue with coins in this description).

The solving of the problem is done by turning the cards with digits and operators and moving them to a line where they should make up an equality.

The Scenario

A sample scenario goes like this:

User starts the program and get automatically the first task with the cards faced down. He decides the level of difficulty he wants to work with and gets a new set of cards. He judges the level of difficulty, the number of cards and decides the amount of time he will need.

He then bets an amount of coins that he can solve the problem in the stipulated time. He turns one card and notices that the odds is decreasing. He bets more coins and decides to start the game without peeping at more cards. The rest of the cards are turned and the time starts running. He starts moving cards in place, but he gets trouble and decides to terminate the problem solving. The coins disappears and the next problem is presented. The user bets again and starts the problem solving. This time he manages to solve the problem within the time limit and the coins multiplied by the odds are added to his coins, and the next problem appears.

The Activity Table

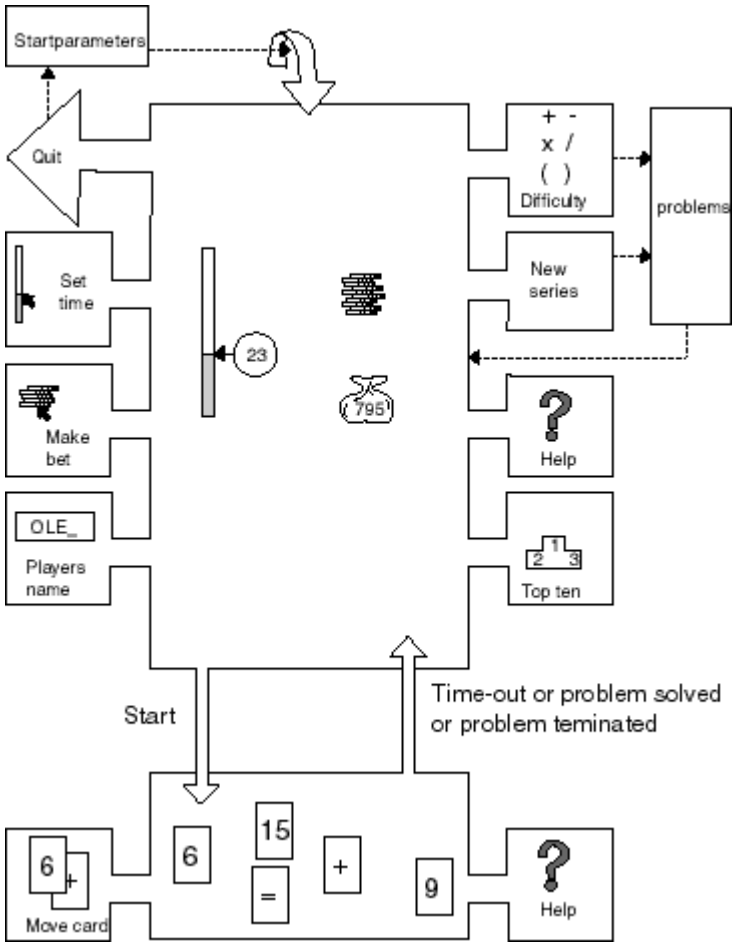
We pick out the activities, the verbs, from the scenario:

The student	The Program	The Teacher
starts the program choose difficulty choose time bets coins turns a card strats the game move a card terminate the problem	shows the problem decreases the odds turn the cards starts clock take away the coins "pay out" the coins	has no spesific task, but set level of difficulty as part of a training session.

The list is not complete, compared to the final functionality of the program, but it focus the important issues in the relation between user and program. It helps us distribute the responsibilities.

The Market

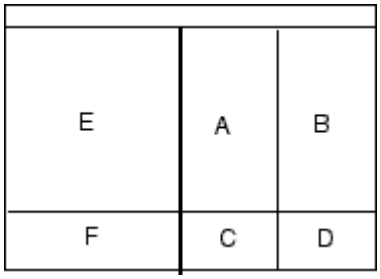
From the anatomy of the program, with a betting phase and a problem solving phase, it is obvious that we will have two main market places. These market places are categorised by the fact that they have different booths associated with them.



Note that the functions help, name registration and a top ten-list based on series of problems are introduced.

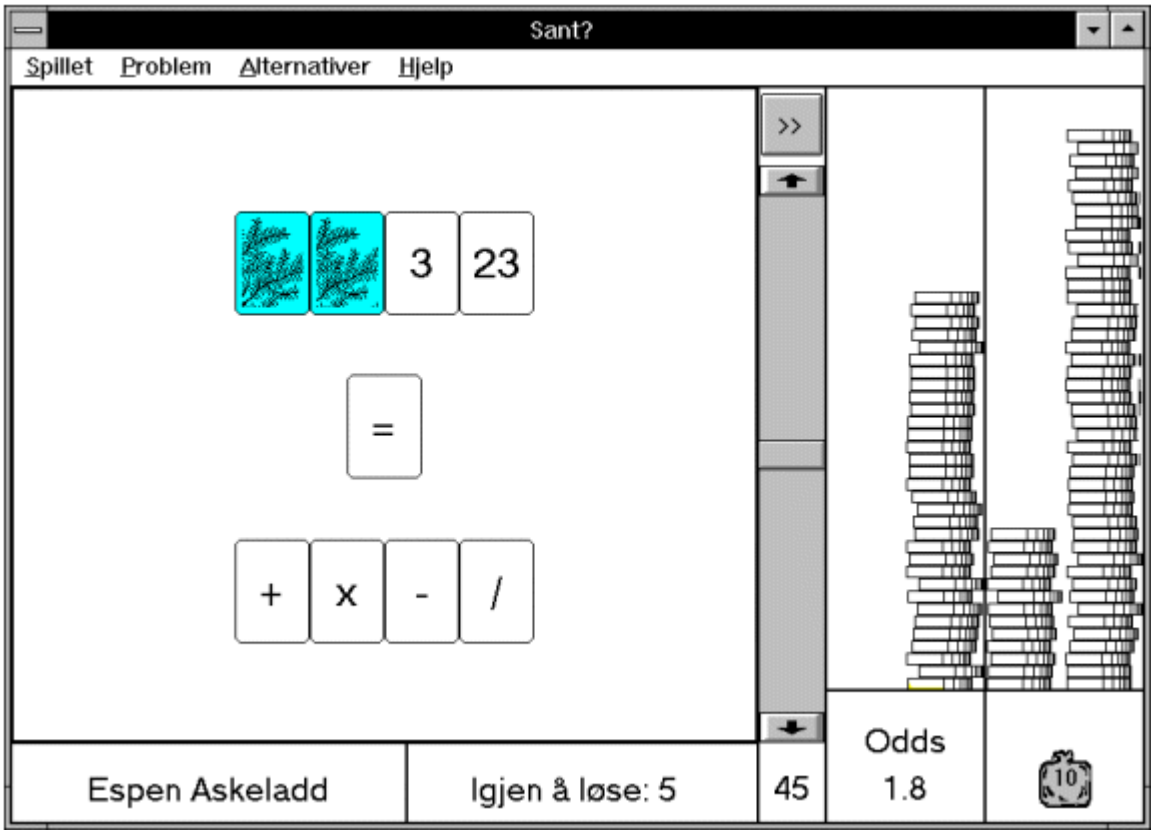
The Screen

The design of the program screen, or window, was focused on keeping the two phases apart and making them clear and easy to distinguish. The main layout of the screen is dividing the area in two parts of roughly he same size. The betting to the right and the problem solving to the left.



A: The betted coins
B and **D**: Users coins
E: problem solving area
F and **C**: Status information

The manifestation of the time was problematic. The chosen solution is a linear time line, which serves as a delimiter between the two areas. The text is in Norwegian, but the screen layout should be self explanatory



(The image has been reduced from the original.)

The Dialog

There are quite a few details in the dialog that had to be resolved. Some of these are:

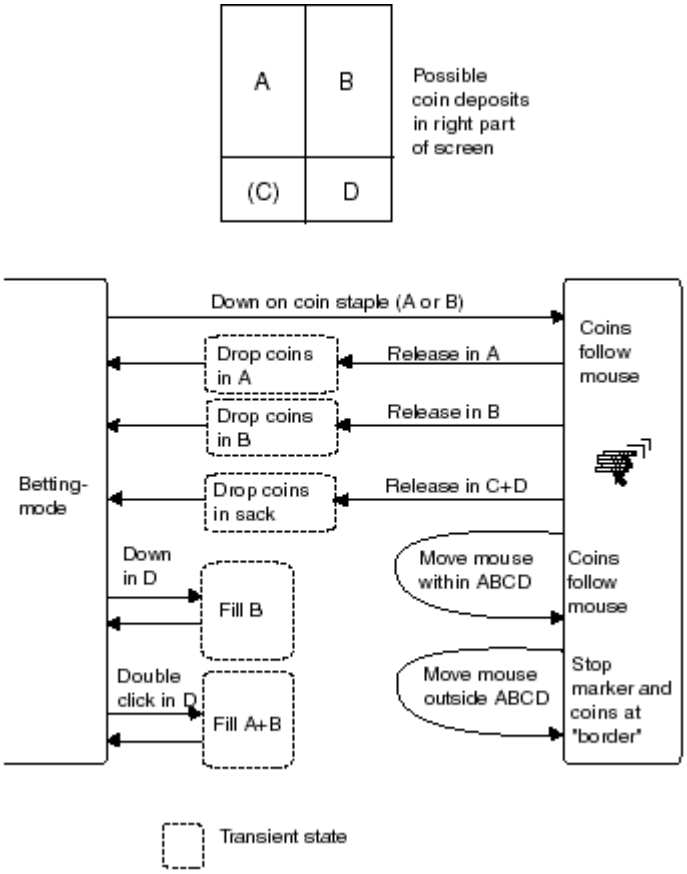
- Coins can be moved by direct manipulation of staples.
- Coins may be freely moved within the three areas: the sack, the users staples (right) and the stake staples.
- A double click to get coins from the sack into the users deposit to the right.
- The time bar acts as a scroll bar when in the betting phase, while the up-functions are blocked during the problem solving phase.
- Multiple ways to start a game: menu, click on the double arrow and turning cards (with a loss of odds)
- Give up a game by moving the time bar to the bottom.
- Turn a card by clicking on it.
- The equation can be built on any of the three lines.
- How much time should we use to pay out and remove coins ? This is a crucial question in a program of this kind. The time and the accompanying (voluntary) sound gives the "beat" to the program.

In general the dialog is based on standard dialog elements and the exceptions, like direct manipulation, is hopefully evident from the metaphor.

The Details

Even if the dialog is fairly simple, there are always some situations that must be considered in more detail. For instance what should happen if the user tries to move coins into the card area and vica versa ? There are a many answers to this and at least a few are aceptable, but we have to decide.

Statediagrams are a useful tool for such detailed analyses. Below is a statdiagram that resolves the move coins problem.



There are a few other situations that have been resolved the same way.

The Program

A demo version of the program can be downloaded here. Some of the funcionality is masked. The text is in Norwegian.

UnPack he zip file, and run Sant.exe.

The full version is available from Nasjonalt Læremiddel Senter.

References

1. The background of the method
   - The Norwegian Ministry of Education launched an effort to work with Computers in Education in the mid-80's. Program design became a substantial part of this. The method is also known as the Grimstad method after the place where a number of design courses where held in the later part of the 80's.
   - The person who introduced the key principles in the design approach was Les Green, a Canadian engineer and remarkable pedagog.
   - The Nordic Council of Ministers launched a similar initiative, and the design approach used in Norway was adapted on a Nordic scale. Some of this activity is documented at Idun, http://www.idun.dk/
   - The deisgn method, together with a number of design issues, is described in more detail in the book (in Norwegian): Brukerorientert programdesign, Minken og Stenseth, 1998, ISBN 82-7726-507-7, Nasjonalt Læremiddelesenter
2. Pedagogy and Technology, A historical Perspective, Børre Stenseth 1999, http://odin.hiof.no/~borres/pedtech/
3. Websters Dictionary, at: http://www.fin.gov.nt.ca/webster.htm
4. Blooms taxonomy, see for instance: http://faculty.washington.edu/krumme/guides/bloom.html
5. The Media Equation, How People Treat Computers, Television, and New Media Like real People and Places, Byron Reeves and Clifford Nass. Cambridge University Press 1996. ISBN 1-57586-052-X.
6. Eliza, Weizenbaum, J., "ELIZA -- A computer program for the study of natural language communication between man and machine", Communications of the ACM 9(1):36-45, 1966. Versions of the program is directly available at numerous web sites, make a web search with "Eliza".
7. Some useful texts on Interface design:
   - Tog on Interface, Bruce Tognazzini, Apple Computer, Inc. 1992, ISBN 0-201-60842-1
   - Human-Computer Interaction, Jenny Preece, Addison Wesley 1994, ISBN 0-201-62769-8
   - The Windows Interface Guidlines for Software Design, Microsoft Press 1995, ISBN 1-55615-679-0.
   - The Art of Human-Computer Interface Design, Brenda Laurel, Addison Wesley 1990, ISBN 0-201-51797-3
   - Apple Computer, Macintosh Human Interface Guidelines, Addison-Wesley, 1992.
8. Ivar Minken, Senior Consultant, Gjensidige Nor, mail
9. Børre Stenseth, Associate professor in Computer Science at Dept. for Informatics and Automation, Ostfold University College, Halden, Norway, mail.
