End-User Development with Design Environments: Gradual Mastery of Programming

Cand. Scient Thesis

Kjersti Sæther

May 2006
Abstract

The end-user development (EUD) community has identified the need for new technologies that support the end user in the development and modification processes of applications. This thesis suggests the use of design environments as an answer to this request. A design environment provides the end users with task-specific programming constructs, while performing design activities. Explanations relevant to the task at hand are provided to the end user as the application is being constructed.

The thesis extends the notion of a design environment further by integrating it into a Flash environment. A prototype demonstrates how an end user may tailor a Flash application at runtime. The tailoring is offered at three levels with increasing complexity (from direct manipulation to programming), which supports a gradual mastery of programming.

The empirical evaluation of the prototype showed that the tailoring of the application could be performed by 1) direct manipulation, 2) integration of new sound, and 3) programming by extension and modification of the functionality. The end users received relevant support to the task at hand, but a tighter integration of tailoring environment and help system is needed. Furthermore, the Flash environment places some restrictions on the technical implementation of this integration, which identifies an area for further work.
**Acknowledgements**

Working with this thesis has been both exciting and challenging, and given me an opportunity to study the effects of end-user participation in the development process, which I find most interesting. I will therefore thank the people who have contributed the most to this thesis.

First of all, I would like to thank my supervisors Anders I. Mørch and Jens J. Kaasbøll. Their productive ideas and remarks, as well as support and motivation throughout the process have been essential to my work.

I will also like to thank InterMedia for providing excellent working facilities. In this context Knut Qvale needs to be mentioned. His technical support in connection with the usability study has been greatly appreciated. I would also like to thank the four participants in the usability study. Without their participation this thesis would not be complete.

Last but not least, I will thank my friends and family for always believing in me. Thank you!

Oslo, 2006-05-01
Kjersti Sæther
# Table of Contents

## CHAPTER 1: INTRODUCTION

1.1 Aim of Study ................................................................. 2
1.2 Scope of Thesis ............................................................... 3
1.3 Structure of Thesis ......................................................... 4

## CHAPTER 2: FROM THEORIES TO CONCEPTUAL MODEL

2.1 Design and Knowledge Creation........................................ 7
2.1.1 Learning by Doing ......................................................... 9
2.1.2 Constructionism .......................................................... 10
2.1.3 Learning by Designing .................................................. 11
2.1.4 Learning by Being Critiqued ......................................... 12
2.2 Design Environments and Tools ........................................ 13
2.2.1 Three Modes of Time ...................................................... 14
2.2.2 Design Environments .................................................... 15
2.2.3 Software Design Environments ....................................... 17
2.2.4 Domain-Oriented Design Environments ......................... 18
2.2.5 Authoring Tools .......................................................... 18
2.2.6 Collaborative Authoring Environments ......................... 20
2.3 End-User Design Activities ............................................. 23
2.3.1 Participatory Design ...................................................... 24
2.3.2 End-User Development ............................................... 25
2.3.3 End-User Tailoring ....................................................... 27
2.4 Conceptual Model ......................................................... 29

## CHAPTER 3: METHODS FOR DESIGN AND EVALUATION

3.1 Research Process .......................................................... 31
3.1.1 Qualitative Exploratory Research .................................. 34
3.2 Design ............................................................................. 36
3.2.1 Prototyping ................................................................. 36
3.3 Evaluation ....................................................................... 38
3.3.1 Evaluation Techniques ................................................ 39
3.4 Summery of Methods ...................................................... 42
List of Figures

Figure 1.1: Scope of Master’s Thesis................................................................. 3
Figure 2.1: Metaphors of learning and expertise............................................. 8
Figure 2.2: Approaches in the knowledge creation process............................ 10
Figure 2.3: Reflection-in-action as a problem solving theory.......................... 13
Figure 2.4: The three components of the Janus design environment.............. 16
Figure 2.5: Tailoring to fit the work situation and resolve design mismatch...... 28
Figure 3.1: The iterative process................................................................. 31
Figure 3.2: Two dimensions of prototyping.................................................... 37
Figure 4.1: Transformation of conceptual model into the prototype.............. 45
Figure 4.2: The metaphors of Flash............................................................. 47
Figure 4.3: Animation in Flash Player 7....................................................... 48
Figure 4.4: Compiler.................................................................................... 54
Figure 4.5: Phases of a compiler................................................................. 55
Figure 4.6: Sothink SWF Quicker 1.7.......................................................... 56
Figure 4.7: Macromedia RoboHelp X5.......................................................... 58
Figure 4.8: The prototype of the integrated design environment.................. 59
Figure 4.9: The prototype................................................................. 60
Figure 5.1: Layout of the experimental setting................................................ 67
Figure 5.2: Location of the property panel in the environment......................... 72
Figure 5.3: Action editor in Sothink SWF Quicker....................................... 75
Figure 5.4: Member list in the design environment........................................ 77
List of Tables

Table 2.1: End-user design activities ................................................................. 23
Table 3.1: The qualitative and quantitative procedure and product .................. 34
Table 4.1: Commands and arguments to FSCommand in Macromedia Flash projectors .... 52
Table 5.1: Classification of participants ............................................................ 68
Table 5.2: Transcription symbol ........................................................................ 70
Chapter 1: Introduction

End-user participation in the software development process is not a new phenomenon, but it has usually been limited to involvement of users in the initial phases of the process (Mørch, 1998). A method that may be applied is participatory design (PD), which is based on the collaboration of end users (or user representatives) and professional developers. Using different collaborative design techniques in the development process can lead to more suitable and durable software applications (Schuler and Namioka, 1993). The more advanced programming tasks are still left to professional developers. However, more of these tasks will be transferred to the end users in the future. This is an assumption made because of increasing numbers of end users, while the numbers of professional developers seem to remain constant (EUD-net, 2003). Methods, techniques and tools applied in this context are referred to as end-user development (EUD). EUD is not a new method; it has existed since the first personal computers became available (Sutcliffe and Mehandjiev, 2004).

EUD is about gaining end user control over certain parts in the development and modification process of applications. It is believed that end users are more qualified to customize the applications to their own work situations. By giving the end user increasing development responsibilities new challenges have arose, which can be summarized as follows: 1) easier programming languages need to be created and 2) the technology must be improved to motivate end users to adopt the technology (EUD-net, 2003). Within this second challenge a third strategy can be identified; designing new technologies with future changes in mind, i.e. designing for modifiability.

Developing programming languages with easier syntax that is adapted to the application domain have been realised to some extent by domain-specific programming languages (DSL). Therefore the focus in this thesis is going to be on the second challenge of developing new technology that motivates end users to do EUD activities. There are several ways to address this challenge, one suggestion is to abstract the functionality and logic of an application onto a higher meta-level. Overwhelming functionality can then be hidden, but still fully accessible for the end users upon demand (progressive disclosure). The end users should even be able to programme, but the focus should not be on mere coding. The meta-level will change the focus from developing by programming to developing by designing. By supporting the end user in design activities and the cooperative problem-solving between user and computer, the
technology will comply with the end users’ needs and not the other way around (Fischer et al., 1989b).

However, to narrow the field even further, another strategy within the challenge will be applied. By designing for possibilities of modifiability and tailorability, end users will be able to further customize their existing applications. The tailoring may be done on three levels with increasing complexity; from direct manipulation to programming. The first level involves changing parameters by selecting among predefined configurations, the second level involves integrating new components into the existing application, and the third level involves programming to change behaviour of the application.

1.1 Aim of Study

The second challenge identified above, is the overall aim of this Master’s thesis and can be summarized as follows:

“…one fundamental challenge of the coming years is to develop environments that allow people without particular background in programming to develop their own applications or modifying existing ones” (EUD-net, 2003)

The challenge is too general and wide to be studied in the context of a Master’s thesis. I have therefore narrowed it down, and identified these three goals:

1. Develop a conceptual model of an EUD environment that supports end users in application-oriented design activities. A conceptual model contributes to understanding the problem space, as well as constructing a blueprint of the overall structure of environment. The environment will support the end users in their pursuit to develop and modify applications without being overpowered by professional developers. This will empower the end users to fully use advanced information systems.

2. Integrate the EUD environment into a Flash domain. By integrating the environment into a Flash domain¹ two things will be achieved. Firstly, the environment can be studied

¹ An environment that is designed for developing and modifying Flash applications, e.g. Macromedia Flash.
within a domain that is often used and familiar to many end users, and secondly, it will limit the scope of the research by addressing a specific problem. In this thesis, it will be further narrowed by focusing on tailoring activities of executable Flash application, i.e. SWF files.

3. **Evaluate the Flash EUD environment with real users.** To highlight problematic areas of the conceptual model, the model must be evaluated. This may be done by use of different techniques. In this thesis this will be done through prototyping, which is a technique for user evaluation. Certain areas of the conceptual model will be turned into a physical design to let end users test the environment. The evaluation will study the tailorability of Flash applications in the EUD environment. The results may contribute to a redesign of the conceptual model and its underlying assumptions so that hopefully, a more user-friendly Flash EUD environment can be developed in the future. It may also contribute to new knowledge in the areas of EUD and EUD environments.

1.2 **Scope of Thesis**

The purpose of this Master’s Thesis is to develop a conceptual model of a user oriented, Flash-based EUD environment, and evaluate it by use of prototyping as described above. To underline this, the chapters of the thesis are presented according to the process of development. The process is as follows:

![Figure 1.1: Scope of Master’s Thesis](image)

*Theories* – The entire research study will be based on theories of end-user activities, EUD environments and learning approaches involved in the process of designing. By combining these theories, an environment can be developed that support the end users in design activities that are essential to improve their work situations.
Conceptual model – The conceptual model is a roadmap of the future EUD environment and means to convey specific ideas to the end users if properly communicated. It is influenced by the theories available, and limited by the affordances and constraints of the Flash environment. The model is developed to test robustness and usability to identify areas of improvement of both Flash and underlying theoretical assumptions.

Design – The next step is to transform certain aspects of the conceptual model into a prototype through a design and implementation phase. The design will have the main focus on tailorable and modifiable in the EUD environment since these are the parameters that are going to be evaluated in the next phase. A physical design of the conceptual model is necessary to give end users the opportunity to test the model of EUD environment.

Evaluation – The design phase is only an incremental step towards the main goal of revealing the weaknesses and strengths of the conceptual model. The evaluation of the prototype will be based on usability of environment, and tailability of Flash applications. It will contribute to new ideas and necessary improvement to the conceptual model, the EUD environment and in some cases adjustments to the existing theories.

The phases described above are more closely examined throughout this thesis and distributed in the five remaining chapters. A layout of the chapters is given below.

1.3 Structure of Thesis

The thesis consists of six chapters together with bibliography and appendix. I will give a short introduction to the different chapters.

This Chapter 1 (this chapter) is an introduction to EUD environments, which entail construction of the conceptual model, design of the prototype, and evaluation of use, along with the description of the aim and structure of the study.

In Chapter 2 the theories that constitute the base of the conceptual model of the thesis are presented and examined. The chapter is divided into three main parts; the knowledge creation process, EUD environments and end-user design activities. The chapter ends by bringing the
theories and ideas together into the conceptual model. The following implementation of the conceptual model is described in a later chapter.

In Chapter 3 the methods and techniques for the design and evaluation activities are more closely examined. The advantages and disadvantages of the design and evaluation techniques applied are discussed, especially, development of conceptual model, prototype, and the usability study. The presentation of these different topics reflects the order of phases in the development process.

In Chapter 4 the challenges encountered when turning the conceptual model into a physical design is described, especially the limitations that the developing tools created. The chapter ends with an interaction scenario of the use of the EUD environment by an end user.

Chapter 5 is the analysis chapter. Data gathered from the observation and interview of the prototype is analysed and evaluated. The chapter starts with a description of the experiment, then the observations and finally the results. Aspects of use observed during the test is analysed and compared to the conceptual model of Chapter 2.

In the final chapter (Chapter 6), summary and conclusions are presented, along with the limitations experienced. Possible directions for further work are suggested to improve the conceptual model and take EUD a step further.
Chapter 2: From Theories to Conceptual Model

The conceptual model is been based on ideas and concepts of what an EUD environment should look, behave and function (Preece et al., 2002). I will therefore in this chapter, present the theories that have influenced the development of the conceptual model. The three main concepts involved are; knowledge creation, design environments and tools, and end-user design activities. I will start by describing process for internalizing new knowledge through design activities. This process is called learning by designing and being critiqued in this thesis. Then the notion of design environment is presented. Design environment incorporate the idea of learning by designing and being critiqued. Finally, end-user design activities that the design environment is intended to support are described.

2.1 Design and Knowledge Creation

The learning paradigm that is applied within the EUD environment is called knowledge creation. It has transcended from two traditional views of learning; knowledge acquisition and participation (see Figure 2.1) (Hakkarainen et al., 2004). Knowledge creation is the transformation of knowledge through innovative activities like designing. Designing is seen as a mental as well as a physical process, with involves creative construction, judgment and dilemma handling (Fischer et al., 1991). It is a problem-finding activity, and therefore the search for new knowledge outside of the conventional educational domain is an important step towards creating new knowledge (Hakkarainen et al., 2004). Traditionally, knowledge was thought to be acquired through mental processes. This does not fit the dual process of design described above. Therefore a new learning paradigm was needed. The knowledge creation metaphor emphasises the adoption of processes, practices and social structures instead of assimilation of current knowledge. This view is a synthesis of more traditional views of learning (Sfard, 1998). The Figure 2.1 below depicts this situation:
The knowledge acquisition metaphor represents a traditional view of learning; learning is the transmission of knowledge from a teacher or advanced student to an individual learner. It is an assimilation of current and existing knowledge (Hakkarainen et al., 2004). In a simplistic view: the mind is seen as a container, and learning is a process of filling the container (Paavola et al., 2002). The participation metaphor however is an alternative approach to the acquisition metaphor. The emphasis is on cultural practices, social interaction, and situated cognition (Muukkonen and Lakkala, 2005). Rather than an individual knowledge formation activity, learning is seen as a process of becoming a participant in the social communication and interaction (Sfard, 1998). “It is more a question of doing things, and participating to expert-like activities, than having knowledge” (Paavola et al., 2003). To participate, the learning can not be separated from the context where it is taking place.

According to Hakkarainen et al. (2004) the knowledge acquisition or participation metaphors do not address the process of creating and advancing knowledge, which is essential for the modern society. The limitations of the acquisition metaphor is based on the common situation that the learner is only expected to assimilate existing knowledge, while the participation metaphor focuses on the adoption of cultural practices and give no special attention to creative changes (Hakkarainen et al., 2004). The knowledge creation metaphor is seen as an innovative process of inquiry where new ideas, tools and practices are created jointly with others (Hakkarainen et al., 2004). The learner’s existing knowledge is either enriched or transformed throughout this process. The EUD environment that was developed with this
thesis tries to support the end user in the pursuit of new knowledge through an innovative activity of design. By allowing the end user to argue about design issues with former and recent users, the current end user may discover new solutions or become able to add or modify their existing knowledge.

2.1.1 Learning by Doing

Even though the concept of learning by doing was introduced long before the knowledge creation metaphor, they may be seen as having some similarities. Both views state that innovative process of creation may lead to new knowledge. Followers of learning by doing believe that knowledge will be created through learning by action, experiment and doing, as apposed to mere thinking. The idea of learning by doing originated from the philosopher John Dewey, as a part of his pragmatic philosophy (Dewey, 1966). He believed that philosophy should be used to obtain concrete actions, and not just answers to fundamental questions. Pragmatism together with interaction, reflection and experience, and the interest in community and democracy, formed what was known as progressive education (Smith, 1997).

Dewey wanted a pedagogical shift from the teacher’s focus on rote learning and dogmatic transmission, towards learning by doing. The style of learning in schools should become more active and self-directed. The method that led to learning was more important than teaching children facts. A famous slogan is: “we don’t teach history, we teach Johnny” (Martinsen, 1991).

The learning by doing theory has inspired other researchers in formalizing their theories and learning concepts, e.g. constructionism and learning by designing. These have again influenced the concept of learning by being critiqued (Figure 2.2). The latter along with learning by designing form the base of the learning approach followed in this thesis, which I call learning by designing and being critiqued.
2.1.2 Constructionism

Constructionism is built on the constructivist view, and they both believe in “building knowledge structures” (Papert, 1991). Even though, both are built on the same assumptions, constructionism is more appropriately classified as an educational method which is based on the constructivist learning theory (Papert, 1991). According to Seymour Papert, constructionism is learning by making. He believed that students will be more deeply involved in their learning if they are constructing something that others will see, critique, and perhaps use. Through construction, students will face complex issues which will lead to problem-solving and increased learning. The motivation for learning originates from the process of constructing (Guzdial, 1997).

Seymour Papert is the founder of constructionism, which is based on Dewey’s principles of learning. Papert’s focus is on the impact of computers in the learning environment. He believes that the technological revolution has brought about a need for improvements in learning, as well as the opportunity to improve the quality of the learning environments. This is the “age of learning” in which our “competitive ability is the ability to learn” (Papert, 1993).
According to Papert (1993), there is a major obstacle to the future of learning, and that is the schools. The schools are stuck in the educational philosophy of the nineteenth and early twentieth centuries. They use a Gothic Cathedral model of learning; a strict plan for placement of knowledge bricks in children’s minds. This traditional view can be placed within the knowledge acquisition metaphor. Instead there should be a construction of a knowledge process from the creation of physical objects. Papert uses an African proverb to describe this situation: If a man is hungry you can give him a fish, but it is better to give him a line and teach him to catch the fish himself. “Children will do best by finding (‘fishing’) for themselves the specific knowledge they need” (Papert, 1993).

2.1.3 Learning by Designing

Designing may be seen as a learning process through problem-finding, which requires feedback and support systems to create new knowledge. Dewey and Papert were primarily focused on individual knowledge construction, without much mentioning of a support system, either by situation or generated by computers. Feedback from teachers is an extremely important and crucial part of learning. It should occur continuously as a part of instruction, but not intrusively (Bransford et al., 2000). By adapting feedback according to the learner’s progress, the benefit will be tremendous. As ICT has become an integrated part of the learning environments, computers have the possibility to take over some aspects of the teacher’s role, e.g. feedback. As a descendent of the theory of Dewey, learning by designing is suggested as the learning approach within the knowledge creation metaphor. This is the approach applied along with a critiquing component in the design environment of this thesis. Learning by designing is inspired by the works of Donald Schön, and I will therefore describe his contributions to this field.

Donald Schön is known for his three great contributions to learning: learning systems, double-loop learning and reflection-in-action. Reflection-in-action (Schön, 1983) is Schön’s contribution to the pedagogical stance of learning by doing. He makes a division between reflection-in-action and everyday action. People carry out many actions, recognitions, and judgments without thinking much about them. Reflection-in-action is closely tied to the experience of surprise. When an action leads to a surprise, either pleasant or unpleasant, the situation starts to talk back. The response may be reflection-in-action: thinking about what is
being done while doing it will influence further doing. If learning is to be achieved, the learner has to enter into a learning process; a process of experiment, situational backtalk, evaluation and reframing (Rømer, 2003). First then, the learner will become a reflective practitioner according to Schön (Schön, 1983).

To be able to give feedback adapted to the learner’s needs and progression, the teacher also has to become a reflective practitioner. As the teacher understands how the learner thinks she may think of new questions, activities and other ways of helping the learner. To achieve this there need to be a shift from “centrally administered, objective measures of student progress, towards independent, qualitative judgments…” (Schön, 1983).

2.1.4 Learning by Being Critiqued

Gerhard Fischer and colleagues take learning by designing a step further by suggesting to implement Schön’s theory of reflection-in-action and situational backtalk by use of computers into a user-centred design environment (McCall et al., 1990). The computers give generated feedback relevant to the task at hand. During construction of an artefact, a situation may talk back to the designer, and a human problem-domain communication will arise. A documenting argumentation or feedback systems (Fischer et al., 1991) will serve as the situation’s backtalk. Documenting argumentation is a kind of design rationale which promotes critical reflection during the design process.

The Figure 2.3 below shows the implementation of learning by being critiqued in a design environment. It supports reflection-in-action by the use of a critiquing component. This is a component that is activated because of a breakdown in the design process, and that helps the designer reflect on the situation. It is explained more thoroughly in section 2.3.2. The designer’s understanding is constantly evolving because of actions that lead to these breakdowns which trigger help and argumentation, e.g. design rationale in the environment.
Chapter 2: From Theories to Conceptual Model

The result of Fischer’s theory has been implemented in the development of Janus (Fischer et al., 1989a), which has influenced parts of this thesis. Janus is a design environment with two critical component for design; construction and argumentation, and with a third critiquing component that combines these. These components will be thoroughly explained in the next section. It is on the basis on the actions taken in the construction of an artefact that the argumentation (or feedback) becomes active. Reflection triggers critics that inform about violation of principles of design (McCall et al., 1990). There is not just violation of principles that may cause situational backtalk, but also situations where no principles exist. Screen dumps of the Janus environment can be seen in Figure 2.4 in the next section.

2.2 Design Environments and Tools

The concept of learning by designing should be taken one step further by incorporating a critiquing component, and materialise the idea by developing a special purpose EUD environment that supports end users with different interests, skills and knowledge in design activities. In the rest of this thesis, these kinds of environments are referred to as design environments.

Fischer has already defined the notion of a design environment, constituting of the three components (construction, argumentation and critics) (Fischer et al., 1989a). Besides describing different design environments, authoring tools that contributed to the implementation is also described in this section. In the end of this section, collaborative

Figure 2.3: Reflection-in-action as a problem solving theory
(Fischer et al., 2002)
Chapter 2: From Theories to Conceptual Model

authoring environments that may be useful to evolve design environments into a collaborative space in the future, is presented.

2.2.1 Three Modes of Time

Development time and use time (or runtime) is familiar concepts or modes when referring to information systems. A third mode is design time, which is highly relevant when design activities are performed in the design environment. Therefore in this context, the three modes are only referring to modes of the design environment and not information systems in general. They should not be confused with other modes of system development activities. The division is made in order to differentiate key activities in design and use of end-user development tools.

**Development time** As the word suggests, development time is the time spent developing a design environment. The functions and features of the environment are implemented during this time. Developing at this stage, may involve extensive programming, and require the use of general-purpose language (GPL) like Java and C++. GPLs are complex languages that are able to address a wide range of problems. Often text-based, GPLs have a syntax and lexicon that have little grounding in the application domain, which may make them difficult to learn and hence employ. Because of their complexity, professionals with programming experience are often the developers at this time, and not end users.

**Design time** It should not be confused with the early stages of a system development. It is here referred to as the time when an artefact or application is constructed or tailored in the design environment. If an artefact is constructed, basic *design units* may be provided to make some parts of the design process easier. To a certain extent end users can be part of this process, but if more advance programming is involved more professional programmers may be required.

**Use time** Modifications to the system can be made during design time or use time. The difference between them is that during use time only minor or temporal modifications to the existing runtime functionality can be done (in principle major changes can also be done, but this is technically difficult to support). Direct manipulation and parameterisation are examples of activities that can be performed at this stage.
2.2.2 Design Environments

A design environment may be developed during development time and usually utilized during design or use time. According to Fischer (1989b), “design environments are computer systems that support design by enabling cooperative problem solving between design and computer.” To achieve this cooperation, design situations need to talk back to the designer. The backtalk is realised through the communication of the components in a design environment. There are two main complimentary components; construction and argumentation, and a third component which link these together; the critics. While constructing an artefact, a breakdown may occur because of violation of design principles which will lead to arguments for improvements in the design. Feedback in this situation may contribute to improving the user understanding. Figure 2.4 shows the three components of the design environment Janus. Janus has incorporated the ideas of reflection-in-action and situational backtalk into the three components; construction, critics and argumentation. The first image of the figure shows the construction area (1) with the critiquing component (2). In this situation, the critiquing component informs the end user of a design violation (2). In this situation, the argumentation provides feedback by explaining the situation and giving possible solutions.
Construction. Within a design environment the construction component is essential. This is the work area where the design activity takes place. The activities may range from direct manipulation to programming, often based on assembling of design units provided by the environment to form a solution that is appropriate to the particular problem domain (Fischer et al., 1998). Activities in this component can include both development and modification of artefacts. In the Janus construction area (1) kitchen design is the main activity. Ready-to-use design units are put together to form a complete kitchen.

Critiquing. Violations in the relationship or breakdowns in the design process may result in messages to the end user, e.g. error messages. This is shown in the first picture (2) in Figure 2.4. The message tells the end user that the refrigerator, sink and stove is too far apart. The component that ties the design units in the construction component and the relationship between them is called the critics or the critiquing component. As in the figure the critics can be visible through a messaging system. However, unlike the message above, information provided by for instance error messages may be incomprehensible. Therefore, within the design environment a feedback component (argumentation) is activated to further explain the situation and suggest solutions. This results in fewer errors, improvement of user’s knowledge and provides a mutual understanding by all participants of the design process (Fischer et al., 1998).
**Argumentation** This component is what I referred to above as the feedback component. The argumentation which this component provides is more than just feedback; it is also a communication between current and previous designers. It is an activity of reasoning about a design. The argumentation activity will include issues, answers and arguments of decisions made during construction (Fischer et al., 1998). By providing pros and cons of following the critics suggestion, the argumentation may help the user understand the consequences of an action (Fischer et al., 1998). The construction component is needed to construct a solution, but the quality of the solution is dependent on the argumentation. The second picture in Figure 2.4 shows an argumentation component. The reasoning for the critique is given in the default viewer (left), together with a correct example of refrigerator, sink and stove distances.

### 2.2.3 Software Design Environments

Another kind of design environment is the software design environment. The various components may not be present or implemented in the same way as described above, but the idea of supporting the design activity through a design-computer communication is present. The main objective with these environments is to support the construction of graphical interfaces by providing ready-to-use design units. Design units may be dragged onto the construction area and linked with other units and existing functionality. The code of the units may be automatically generated whenever desired. This may replace some of the tedious and difficult tasks of coding all aspects of an application. The advantage of this approach is that applications can be more easily and rapidly modified, tested and reused. There are several software design environment on the marked today of which I will mention two.

**VisualAge for Java** This supports complete development of applications by use of Java programming. VisualAge for Java support both visual programming and manually writing Java code. The visual programming activity involves the use of components like JavaBeans. They are reusable software components that often hide complex, internal functionality for the user. In this software environment the beans can be dragged and dropped onto the Visual Composition Editor and linked together to make the application. It is also possible to connect visual components with non-visual components (Akerley et al., 1999). Even though the design mostly involve visual components, it may be useful with some programming experience.
**Visual Studio .NET** Another example of a visual programming environment is Visual Studio .NET. As opposed to the Java environment, Visual Studio .NET supports the development of Windows and web applications. This environment is associated with Microsoft and supports programming in Visual Basic, C++ and C# among others.

### 2.2.4 Domain-Oriented Design Environments

Another form of design environment is the domain-oriented design environment (DODE). A software design environment may be quite general, but by making the environment domain-oriented, the user can concentrate on their area of expertise. It will also give feedback that is more appropriate for the situations.

“To provide the user with the appropriate level of control and a better understanding, we have to replace human-computer communication with human problem-domain communication, which allows users to concentrate on the problems of their domain and to ignore the fact that they are using a computer tool.” (Fischer and Lemke, 1988)

Shifting the focus from human-computer communication to human problem-domain communication may reduce the conceptual distance between problem-domain semantics and software artefacts (Fischer and Lemke, 1988). An example of a domain-oriented design environment is Janus (Fischer et al., 1989a) that attempted to bridge this distance (see Figure 2.4). This is a domain-oriented design environment for kitchen design. The critics and feedback are concentrated about the design of kitchens, in an attempt of a human problem-domain communication. If design principles are violated, e.g. the distance between stove and sink, the critiquing component is activated and alerts the argumentation component that will suggest different alternatives to solve the problem (McCall et al., 1990). Janus therefore provides the designers with information about principles of the design and the reasoning underlying them at the same time as constructing the artefacts.

### 2.2.5 Authoring Tools

In contrast to design environments that support the user throughout the entire design process, an authoring tool has usually one objective, and that is to produce content. The concept of
authoring tool is presented here because a content authoring tool was used in the development of the prototype, as well as utilized as a construction component in the design environment.

Authoring tools are often used to create professional, engaging and interactive content without the need for programming or the assistance of programmers. In order for an authoring tool to be successful it must be possible for end users to have as much access as possible to the technology without requiring the assistance of technology experts for doing basic authoring activities (Piguet and Peraya, 2000). Today there are hundreds of different tools on the marked that specialize in the creation of different content for different applications, such as e-learning. There are web site authoring tools that help the user build and link individual web pages to create a web site, testing and assessment authoring tools that measure the effectiveness of learning, help authoring tools that help create online help files, course authoring tools that help creating e-learning courses and collaborative authoring tools that aid users create content together (Grøva, 2004).

The reasons for using authoring tools may vary, some want control of artefact and independence of programmers, while others want to reduce costs of hiring consultants by in-house development. Independent of motives, the tools need to support the user whatever level of expertise she possesses. Limitations in the tools become apparent when looking at the two conflicting factors; ease-of-use and expressiveness. Few authoring tools are able to incorporate both factors. At one end there are tools that are template-based and need little or no training. These have limited flexibility and functionality, but are easy-to-use. At the other end, there are authoring tools that are designed to produce high-end multimedia simulations. These tools require extensive training, but offer a great deal of creative freedom. Most of the authoring tools in the market are in the mid range. They require some training, but offer creative freedom. The use of the authoring tools gives no guarantee for high quality applications. End-users are often not aware of design rules to increase the usability of the application, and lack technical insight to make the application reusable according to standards (Preclík, 2000).
2.2.6 Collaborative Authoring Environments

Collaborative authoring environments are an extension of authoring tools with a collaborative feature. They are described here, because as emerging technologies they can contribute to improve the collaborative aspects of design environments, e.g. collaboration between different designs and their creators in the argumentation component. As opposed to ordinary authoring tools, these are environments where users work together to create content. Weblog and Wiki are good examples on emerging collaborative authoring tools, also referred to as social software. They are seen as lightweight tools and make information highly visible.

Some may argue that social software has existed for a long time, e.g. e-mail and message boards. But what differentiate these from the Wiki and Weblog is that fact that these are top-down approaches, compared to social software that are bottom-up (Boyd, 2003).

Weblog are frequently updated websites composed of automated reverse chronological posting functions. It is an effective way to share interests, information and opinions. The content can either be personal or more professional oriented. To the average blogger, the Weblog is used as a personal diary. A growing number of Weblogs are being used by professionals as personal knowledge repositories, learning journals or networking instruments (Efimova and Fiedler, 2004).

In the beginning the Weblogs were created by Web professionals, or others that had some knowledge of HTML (Blood, 2002). New bloggers have little or no technical background, and may find it difficult to set up a Weblog. The average Weblog tool works as a lightweight content management system. It consists of a database of text entries and other pieces of content (images, sound, etc) and supports adding and editing of items by providing pre-defined templates and step-by-step instructions (Efimova and Fiedler, 2004). Examples of well known Weblogs are: Blog*Spot, Blogger and MSN Spaces. Grudin (Grudin and Poltrock, 2005) has categorised the different blog types:

- **Diary-like blogs**, the most common type of blogs. It is authored usually by a young person primarily to be read by friends and family.
• *A-list blogs*, written by journalists or others. They are a good source of information on events, products and trends.

• *Watchlists*, reports on appearances of a topic (text string) in any blogs. This is a powerful way to be up-to-date on topics being discussed around the world.

• *Externally visible employee blogs*, typically discussing both personal and work life. This may put a human face on an organization or product.

• *Project blogs*, authored by multiple team members with the main focus exclusively on work. This is internal equivalent to the externally visible employee blogs.

Using blogs as collaborative tools in learning environments have shown an increase in learning outcome (Du and Wagner, 2005). The instant feedback is an effective tool that encourages “conversation” among learners. Other reasons may be (Efimova and Fiedler, 2004):

• *Learning from multiple perspectives* – allow learners to go beyond group thinking by supporting diversity and bringing together multiple perspectives and backgrounds.

• *Synergies of self-organised and community learning* – a personal Weblog do not impose any communal learning agenda and learning style, at the same time learners can benefit from a community feedback, validation and further development of ideas.

• *Digital apprenticeship* – regular reading of other blogs, provide novices with opportunities to learn from experts regardless of geography and disciplinary boarders.

• *Support for the development of meta-learning skills* – inner conversations and reflective thinking become available for review and development through the blog, which promote better skills for learning.

*Wiki* was first coined in 1995 by Ward Cunningham, as a medium where one could communicate easily and asynchronously with site users on different topics. As opposed to a blog, anyone can edit an existing Wiki page; it is accessible and editable by the site users. The
open principle, which allows complete freedom to revise any aspect of the webpage, may make some users apprehensive (Gonzalez-Reinhart, 2005). The content is however, protected against mistake or malicious editing by revision control which means that old pages can be restored.

The reverse chronological listing of posts in a blog may be a drawback for certain types of information (Dickerson, 2004). The lack of structure in a Wiki, allowing information to be arranged in any way that makes sense without restriction solves the problem. Despite this, there have been registered some disadvantages that has been solved as following (Salustri, 2005):

- **Revert war** – a contributor with strongly held opinions has the possibility to continually “revert” Wiki pages back to previous versions to fit her views. This has led to the Wiki etiquette rule of avoiding making categorical statements.

- **Refactoring** – Wiki pages can quickly be disorganised when more and more contributors add content. This has given raise to refactoring. Original commentary and discussion are not deleted but placed in an associated Wiki page and linked back to the original. This gives the user the option of reviewing the original discussions or simply read the summary and results.

- **Pull technology** – Wiki is a pull technology instead of a push. To see resent changes and additions, the user need to intentionally visit the Wiki page. This limitation can be addressed by implementing a notification service when new additions are made.

The usability of a Wiki is meant to be easy, and should not demand much technological knowledge. Nevertheless, few Wiki sites uses WYSIWYG (What You See Is What You Get) editor to facilitate easy editing, instead simplified mark-up language is used that may be difficult for non-technical experts. In fact, a usability test of Wikis showed that non-technical users find the creation and management of links to pages and images quite difficult (Désilets et al., 2005).

Wikis have been used to collaboratively create and maintain software documentation as frequently asked questions, textbooks, travel guides, repositories and specialized knowledge
bases. The best known example is probably Wikipedia which is an online encyclopaedia (Wikipedia, 2006).

2.3 End-User Design Activities

Within design environments different end-user design activities can be performed. Their impact may vary, and not all exiting end-user design activities are relevant as activities performed in the design environments. However, in this section I am going to look closer the activities where end users are involved, especially those within the field of participatory design (PD), end-user development (EUD) and end-user tailoring (EUT). As an attempt to differentiate between these three groups, I will try to categorize the different PD, EUD and EUT activities according to the degree of end user’s impact on software artefacts. The result of this is presented in Table 2.1.

![Table 2.1: End-user design activities](image)

Table 2.1 displays PD, EUD and EUT design activities placed within three categories of non-computing, direct manipulation and programming. In the non-computing category activities are constructed without the use of computers, in cooperation between user and professional developers, while in the direct manipulation-programming category the activities ranges from selecting among pre-defined configuration to general-purpose programming. The direct
manipulation-programming dimension is a gradual increase in complexity of development and learning involved.

Participatory design activities are mostly present in the requirement phase, where the end user works in close collaboration with professional developers. These activities involve modifying or developing often visual artefacts that may not be part of the final product. However, the end user’s direct impact on the final product increases by use of cooperative prototyping (Bødker and Grønbæk, 1991). End users have the possibility to modify the prototype by direct manipulation, but if more advanced techniques are needed, like programming, professional developers will have to get involved (Bødker et al., 1993). The main difference between PD and EUD or EUT is that EUD and EUT activities are used in the later stages of the design process, often when a system is up and running. EUD is in this thesis is first and foremost seen as a way to create, but also modify software artefacts. It often involves visual programming, which is programming by use of visual representation (Cypher et al., 1993). Programming by demonstration, programming with example and macro generation may all be seen as forms of visual programming, but with increasing end user impact on the artefact. Instead of using text as in general-purpose languages, images are used that generate the code needed without the end user ever seeing the written code (Cypher et al., 1993). EUT can be seen as a modification activity that covers the activity from parameterization or customization to programming by use of general-purpose languages. PD’s cooperative prototyping and EUT’s customization have been placed at the same level of impact because direct manipulation is involved in both activities. This is also the case in parts of EUD’s visual programming and EUT’s integration, as well as EUD’s scripting and EUT’s extension because of similar techniques may be used in both design activities.

2.3.1 Participatory Design

In the initial design phase, professional developers and end users may collaborate to develop a set of requirements for the system. This collaboration may be part of an approach called participatory design that was first applied in the Scandinavian projects of DEMOS and UTOPIA (Nygaard, 1986; Bjerknes et al., 1987). The motivation for this was political as well as technical (Ehn, 1992). By giving every employee an opportunity to take part in the decision-making that has impact on their working situation, the democracy, power and control at the workplace may be increased (Bjerknes and Bratteteig, 1995). PD may also
Chapter 2: From Theories to Conceptual Model

contribute to more successful design and high-quality products, as well as increasing the system’s knowledgebase among the users, create realistic expectation of the system, and reduce the resistance to change (Bjerknes and Bratteteig, 1995). However, these attributes are not just unique to PD, they may also apply for other user participation approaches.

Since PD is usually performed in the requirement phase, a set of methods and techniques are applied to elicit user requirement, which the professionals later may incorporate into the development phase. The activities involved are used to provide developers and end users with a common media for communication and an opportunity to learn each others language-games (Ehn, 1992). A language-game is the language used within a profession (Ehn, 1992). Learning another language-game involves following the rules of the game, i.e. act in a way that can be understood by other participants in the game. The PD activities are therefore important in the process towards learning each others language-games.

Low-fidelity prototyping, mock-ups and scenarios are examples of activities often used in PD. These may be used to imitate a system, individual screens or other artefacts, and are often made up by cardboard, wood or paper. Their intention is to mirror reality and act as remainders of earlier experiences (Ehn, 1992). Because these artefacts are simple, cheap and quick to make, they are also simple, cheap and quick to modify which may be beneficial when testing alternative ideas and designs (Preece et al., 2002). Another activity within PD is cooperative prototyping (Bødker and Grønbæk, 1991), which may be seen as a high-fidelity prototype. In Table 2.1, this activity is separated from the other PD activities and overlaps with EUT’s customization. This is due to the fact that the end users have the possibility to direct manipulations to functionality of the cooperative prototype (Bødker and Grønbæk, 1991).

2.3.2 End-User Development

PD is often practiced in the initial phases of software development and ends after installation, while end-user development (EUD) activities may also be practiced during runtime and give the user the opportunity to create or modify software artefacts without interference of professional developers. Before describing the EUD activities, the approach should be
The definition of EUD is as stated by Network of Excellence on End-User Development (EUD-net, 2003):

“End-user development is a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professionals software developers, at some point to create or modify a software artefact”.

End users are knowledgeable workers of an application domain that given the right methods and tools should have the ability to create or modify an artefact according to their own preference and needs. Tasks that were traditionally performed by professional software developers may be transferred to the end users. To support these tasks, environments that support the perspective of easy-to-develop need to be build (EUD-net, 2003). According to EUD-net (2003) developing design environments and tools that support end-users, who do not have a particular background in programming, to tailor and even develop their own applications is a challenge within the EUD community. This is an approach that is explored through this thesis.

In the field of EUD there are several areas of focus, e.g. programming paradigms and languages, methods, environments and tools, architectural issues, interaction techniques, application domains, and organizational and social issues (EUD-net, 2003). Going into all these areas would be too extensive for this thesis. Therefore, the main areas of interest for this thesis are environments and tools, and EUD activities. Costabile (Costabile et al., 2003) divides the end user activities into two groups:

1. **Parameterisation, customisation or personalization.** These are activities where the users can choose among alternative behaviours that are already available in the application.

2. **Programming.** This activity will result in a modification or creation of a software artefact. The programming paradigms used may be programming by demonstration, programming with examples, visual programming, macro generation and scripting.

According to EUD-net (EUD-net, 2003), the first activity is not an end-user development activity because it only involves the use of an application. There are no real modifications
performed since information is entered according to predefined configurations (EUD-net, 2003). The second activity is however, considered as an end-user development activity. In the Table 2.1, the different programming paradigms are distributed within the direct manipulation-programming dimension of the end-user design activities. This is not an exact ordering of the activities, rather an attempt to separate them. The first paradigm in the table is visual programming, which is the opposite of textual programming that use traditional languages. The words are replaced by images that represent the syntax, which make it easier to understand and develop (Kowalczyk, 1997). Macro generation is a program that records a set of user actions, to simplify repetitive tasks. Spreadsheet programs may often have this functionality. If the tasks are not identical with the original recording the recording is useless (Kowalczyk, 1997). Programming by demonstration is an extension of macro recording (Kowalczyk, 1997), and may be applied in tasks that are similar and not just identical. By providing examples of different tasks the program should infer the user’s intent, and provide a more general code (Cypher et al., 1993; Kowalczyk, 1997). In accordance with Cypher (1993) programming by demonstration and programming with examples are seen as one and the same activity in this thesis. All these paradigms are mostly developed on the basis of direct manipulation. Further down this dimension, model based approaches like Unified Modelling Language (UML) (OMG, 2004) are placed. The visual modelling language permits the end user to make an abstract model of an entire system by use of diagrams. However if desired, programming may be involved at this stage. The last EUD activity is scripting, that is extended to object-oriented scripting. It may be seen as a simplified programming language where the syntax and vocabulary are somewhat similar to the users’ natural language (Kowalczyk, 1997). In comparison to the other EUD activities, this is text based and involves writing code, and is therefore placed even further down in Table 2.1. The EUD activities are mostly at the same levels as the end-user tailoring activities described in the section below. One exception is the activity of parameterisation, customization and personalisation which are considered as an end-user tailoring activity, but not an EUD activity.

### 2.3.3 End-User Tailoring

End-user tailoring (EUT) covers the activities from customization to programming by use of general-programming languages (GPL). Tailoring enables the end user to modify existing system functionality of the software application at runtime as opposed to modifying it during
development. Thus, the tailorable application needs to be designed to support tailoring opportunities and unanticipated use. Figure 2.5 shows the tailoring activity before and after the modifications (Mørch et al., 1997).

![Figure 2.5: Tailoring to fit the work situation and resolve design mismatch. This is also referred to as adaptive EUD (Mørch et al., 1997; Mørch, 2003)](image)

The left side of Figure 2.5 depicts a breakdown that may occur when there is a mismatch between the application and the work situation (Schön, 1983; Winograd and Flores, 1986; Fischer, 1993). This may lead to the need for tailoring the application to fit the work situation. Since end users are experts on their own work, they are better fit to modify their own applications.

The tailoring process is divided into three end-user design activities according to degree of difficulty and impact on application (Mørch, 1998). The first level is the customization activity. At this level the end user is allowed to tailor the interface by modify parameters of already existing application components. Customization is defined as follows:

“Selecting among a set of pre-defined configurations. This can be done directly by modifying the appearance of presentation objects, or indirectly by setting parameters (attribute values) in property sheets or from menu options” (Mørch, 1995)

The customization level requires the least technological knowledge and experience, and is therefore easy to use and understand. Modification at this level involves direct manipulation to change the appearance of objects, i.e. resizing and replacing pictures or setting parameters. Since there is no programming involved, the changes that can be made are limited. Integration activity is at the second level of EUT (Table 2.1). It surpasses customization by
allowing new functionality to be added to the application by creation of macros, script recordings or by advanced copy and paste functions. New functionality is added by linking together predefined components within or between applications (Mørch, 1995). Integration is further divided into hard and soft integration according to the degree of modification involved. In hard integration a new component is created that is attached to the original application. Soft integration however, integrates program executions such as function calls and objects instances, or program documentations. Extension is the third level. It allows more radical changes to the application:

“Extension is the approach to tailoring where the implementation of the application is improved by adding new code.” (Mørch, 1995)

Some changes cannot be anticipated by the developers at design time, thus changes must be made to the existing code by adding new code. The grade of complexity decides whether these changes need to be made by an end user or a professional developer. There is no requirement of which type of languages to use to add code. Scripting as a EUD activity has therefore some overlapping qualities with extension. However, Mørch (Mørch, 1995) argues that it should preferably be in GPLs to be able to use sub-classing and method extension, in order to make radical changes to computer applications.

### 2.4 Conceptual Model

From the theories presented in the previous sections, a mental model of how a design environment may look, behave and function has emerged (Preece et al., 2002). The idea of the future design environment may be seen as blueprint for the design that is presented in Chapter 4. By transforming the mental model into a physical object through prototyping, the idea of the environment may be more easily conveyed to the end users, and new requirements and needs may be identified (Preece et al., 2002).

The conceptual model is made up by three main units that contribute to the focus of the thesis, as well as restricting the model. The ideas and concepts of the conceptual model are as follows:
Design environment:
- Construction
- Argumentation
- Critics

End-user design activities:
- Customization
- Integration
- Extension

Learning by designing and being critiqued

One of the restricting factors is the specification of the three components (construction, argumentation and critics) of the design environment. They contribute to forming a specific framework of the environment. There are however, few requirements on how to implementation them, except for the underlying learning theory of learning by design and being critiqued. This imposes a step-by-step learning process on the activities performed within the environment. Another restricting factor is the modification activity (customization, integration and extension). This is however, a voluntary restriction because of the aim of the thesis.

Conceptual models may often be evaluated through the use of non-computing devices like mock-ups and sketches. However, in this study a prototype containing most of the design environment’s features and some functionality was applied. This was done to make the evaluation as realistic as possible.
Chapter 3: Methods for Design and Evaluation

The research study was based upon the iterative processes of conceptual model, design and evaluation (Figure 3.1). The conceptual model presented in the previous chapter was transformed into something physical, i.e. the prototype, and evaluated by end users. The evaluation focused on the possibilities of end-user tailoring activities in the environment. The result of the evaluation may later contribute to the redesign of the environment, and adjustments to the conceptual model. In the end a more usable design environment will hopefully be the outcome.

![Figure 3.1: The iterative process](image)

In this chapter, the conceptual model, design and evaluation methods applied will be described and justified.

3.1 Research Process

Qualitative and quantitative research studies are the two main approaches applied in evaluation. It may however, be difficult to be categorical when defining the two approaches. The literature is full of various classifications, and disagreements of their belonging techniques flourish among various opponents. I have therefore tried to give a fair representation of the approaches, which may still be criticized.

To make the decision between a qualitative or a quantitative research study, the researcher need to be conscious of what she wants to find out (Silverman, 2003). If the researcher wants to find out how people vote in an election, a quantitative approach may be appropriate. On the other hand if the researcher wants to find out why people vote the way they do a qualitative approach may be more fitting.
However, there are some guidelines that can be followed. If there is no clear defined problem or scope and further insight is desired, an exploratory research may be useful. If however, decision-making and conclusions are needed conclusive research may be preferred (Joppe, 2006). The techniques applied in either case may be said to be more or less qualitative or quantitative in nature. For example, qualitative research will often be exploratory or initial research (piloting) before conducting more conclusive, quantitative research (Joppe, 2006). Newman and Benz (1998) agrees with this view of a continuum between the approaches by saying “…what are known as qualitative methods are frequently beginning points, foundation strategies, which often are followed by quantitative methodologies” (Newman and Benz, 1998). However, this is only suggestions and guidelines; exploratory research can therefore be quantitative and conclusive research qualitative (Joppe, 2006).

To summarize the different data and design collection techniques, as well as the evaluation techniques available and the respective end products, I have constructed the Table 3.1 shown at the end of this section. This is an attempt to separate the qualitative and quantitative approaches according to procedures applied and results produced.

**Procedure** Within the research process, the procedure is defined as the data collection techniques and evaluation process. The most common data collection techniques used in qualitative research are; observations, interviews and document analysis. They may be presented in different kind of media like audio, e-mails, video and images etc. (Creswell, 2003). A quantitative approach on the other hand, often uses methods like surveys, laboratory experiments, or formal and numerical methods (Newman and Benz, 1998). Another way of differentiating according to procedure is the scope of the research. The qualitative researcher usually goes into the depth of a research field rather than the width like in a quantitative study. This means that the researcher may be examining one instead of many settings, and that the setting is seen as a whole instead of abstracting a part of it (Repstad, 1998). In a quantitative study randomization, control of variables, and valid and reliable measurements may be required, and the aim is often to generalize from the sample to the population (Newman and Benz, 1998). Yin (2003) calls this statistical generalization, as opposed to analytic generalizations used in some qualitative studies.

However, the categorizing of procedures into qualitative or quantitative may lead to misinterpretation, because of similar techniques used, e.g. observation (Silverman, 2003). It
may therefore be difficult to differentiate the approach according to techniques alone, but can be revealed by looking at technique and research scope in combination (Silverman, 2003). The scope of this thesis was to gain insight into the end-user tailoring activities in a design environment. Therefore observation of end users’ activities was favoured. As can be seen in Table 3.1, observations can be both qualitative and quantitative depending on the character of the observation, e.g. structured or unstructured, and the desired end product (Silverman, 2003). However, the use of observation as a way of collecting data was not meant for statistical analyses of the number of pages visited or functions used, but merely a way of analysing the use of the environment. The procedures used in this thesis may therefore be seen as qualitative.

**Product** A more visible and definite division between these approaches can be seen in the end product. The result of the evaluation either statistical or textual analysis, may result in a numerical or textual product. In a quantitative research study, numbers play an important role. Numbers are used to statistically describe the distribution of phenomenon, comparisons, and correlations and representativness of the sample (Repstad, 1998; Joppe, 2006), while in the qualitative approach, text is the means of expression. This does not mean that only texts are studied, also other media can be used. Data are however usually recorded textually and analysed on the basis of the text (Repstad, 1998).

Because of the nature of procedures applied, no statistical analysis was needed, thus no numerical product was generated. The final product of the evaluation was described textually.
3.1.1 Qualitative Exploratory Research

On basis on the former description of the different approaches this thesis followed a qualitative exploratory approach. Since the principal motive at this stage was evaluating the design environment and get insight into the use of the environment, generalization of results was impossible and not wanted at this stage. To collect information about the usability, a usability study with the think-aloud technique was conducted.

When doing qualitative research, there are some pitfalls that are important to be aware of. Some may see qualitative research as an interpretive activity: “One cannot escape the personal interpretation brought to qualitative data analysis” (Creswell, 2003). In this study this applied to the data that were collected through the usability study. The data were filtered through the participants’ worldview, and then analyzed through my personal filters. “Although data collection may be convenient and easy, the problems of reporting data that are biased, incomplete, or compromised are legend” (Creswell, 2003). By using different data collecting techniques and cross-checking which may be referred to as triangulation, the validity of the findings may have increased.

**Triangulation** To help the researcher understand a problem or situation better, different perspectives may be used (Preece et al., 2002). This can be obtained by using different data gathering techniques like interview, observation and documentation. Weaknesses in one
technique may be compensated by the use of another. In this thesis, the results from observing, interviewing and testing users were triangulated. The main objective of the usability study was to observe the users while performing end-user tailoring activities. The three tasks that the participants were to perform covered the activities from direct manipulation to programming. Each session was video taped so every movements and comments made by the users could later be analysed. Before each session each user were asked of their programming skills and general ICT background. After the session, an informal interview was conducted where the user got the possibility to comment on their performance. The latter techniques supplemented the main activity of observing.

**Sample size** Another topic that needs to be addressed is the number of participants compared to the total population. In quantitative research the sample size and randomization of the sample may be especially important. This is because the sample should be a reflection or representation of an entire population. In this qualitative study however, sample size and randomization was not paramount. Nevertheless, there are recommendations of sample sizes when testing interactive systems. Some say that the sample should be between five and twelve (Dumas and Redish, 1999). Others states that as soon as the same kinds of problem starts being revealed, its time to stop the test (Preece et al., 2002). According to Nielsen (2000) this will happen after observing the fifth participant in a usability study, and the sample size in a usability study should therefore not include more than five participants. This conclusion was drawn from a statistical formula that calculate the number of participants needed in a usability test (Nielsen and Landauer, 1993). Even though, there have been some discussions about this sample number and the underlying assumptions (Woolrych and Cockton, 2001), I had decided to follow the recommendations from Nielsen (2000). At this early stage of system development finding as many user problems as possible were desired, and not a representative usage. To be able to study the aspects of use, the composition of participants was more important than the number of participants.

Participants with different backgrounds and skill levels were asked to take part in the study. Because of time constraints and convenience of this study, flyers were only put up at the University campus. Incentives were offered, but unfortunately no one volunteered. It was therefore necessary to ask specific students. To get a diverse sample as possible I contacted students with various ICT skills, from expert to novice. A division of participants according to programming skills and Flash experience can be viewed in Table 5.1, Chapter 5.
3.2 Design

Since the framework of the research study has already been described, the actual techniques involved can now be more closely examined. The conceptual model was the starting point for the design process. The model was based on an understanding of the theories and design problems involved that made up the problem domain (Preece et al., 2002). The prototype was then developed from the ideas of the model. Because of few user requirements available for the development of a design environment, the prototype had to be based solely on the theories presented in the previous chapter. Besides this, the technological affordances provided by the Flash environment had to be studied.

The theories and ideas were based on the information gathered from written data sources like articles and books, published and unpublished documents, reports, email messages, manuals, reference guides, and web-based help system (FAQ) and news groups. Especially, some of the latter data sources have been important in the search for solutions within the field of Flash environments, where there is little established research literature available. The validity of the written text must be considered before it is used in any research study. Different authors may have different motives for writing the text. For instance, a leader in an organization will often have a different view than an employee on the same matter (Repstad, 1998). Thus, when using Internet sources as in this study, the researcher needs to be specially careful in examining the motives, especially in the cases where the author is unknown.

To get appropriate feedback on the conceptual model and design choices, prototyping can be a valuable technique (Preece et al., 2002). Therefore, the second step in the iterative process was the development of a prototype. A prototype is a communication device between the users and researcher (Preece et al., 2002). There are several prototyping techniques, and different types are used in different contexts. I wanted to develop a prototype that seemed as real as possible, and displayed the possible features of a design environment.

3.2.1 Prototyping

There are two main types of prototyping that are important within the exploratory field of interaction design; low-fidelity prototyping and high-fidelity prototyping (Preece et al., 2002). Some researchers also include a third type; medium-fidelity prototyping to differentiate the
degree of accuracy in representation between the different types (Greenberg, 1998). In this thesis however, I will only separate between high and low-fidelity. Low-fidelity prototyping was classified in the previous chapter as a participatory design activity. This approach is mainly used in the early phases of the design process, to test conceptual models, design alternatives and screen layouts (Rudd et al., 1996; Svanes and Seland, 2004). The functionality is non-existent or limited, and bears little resemblance to the final system. It may be made out of paper, cardboard or wood. Storyboarding and sketching are examples low-fidelity prototyping (Preece et al., 2002). High-fidelity prototype however, is not as quick and easy to develop as a low-fidelity prototype. It may be more faithful to the intended interface, and the resemblance to the final product may increase (Rudd et al., 1996). Therefore it may offer a higher degree of interactivity and realism during tests and evaluations. However, the code beneath the surface may be incomplete and quite simple. To let the end users test the features of a design environment a high-fidelity prototype was developed.

High-fidelity prototyping can further be divided into vertical and horizontal prototyping. These two approaches limit either the prototype’s functionality or features. Choosing one over the other entails a trade between features vs. functionality as shown in Figure 3.2 below.

![Figure 3.2: Two dimensions of prototyping (Greenberg, 1998)](image)

A system may be seen as consisting of several layers from user interface to the operating system. In horizontal prototyping only specific layers are implemented with seemingly complete features, e.g. a user interface. In a vertical prototype however, only selected features of the target system is implemented completely throughout the functionality (Budde et al., 1991).
I chose to implement a horizontal, high-fidelity prototype. It gave the users a possibility to evaluate how the real environment would function, along with the ability to study different features like editing, coding and use of help system. However, the entire functionality was not present, only the illusion of it. Certain paths were developed so the users would be able to perform the tailoring usability tasks. The paths contained some shortcuts, but tried to limit any negative influence on the problem solving process. In those cases it influenced the participants’ performance it is taken into consideration in the evaluation phase.

3.3 Evaluation

The last stage of the iteration process before redesign is evaluation. Evaluation of a system is the process of collecting data of a particular group of users, using an application in a certain type of environment (Preece et al., 2002). There are however, three questions that need answers before starting on the evaluation process (Preece et al., 2002):

1. *What to evaluate* – On the basis on the conceptual model, a prototype of a design environment was realised. The aim of the prototype was to implement a design environment integrated with a Flash environment. The environment should be able to support the end user to tailor a Flash application at different levels of complexity; from direct manipulation to programming.

2. *Why to evaluate* – In a user-centric development process, it is important to get the user’s opinion of the design environment. However, at this stage of the development process, any statistical analysis of the user performance was not wanted. Rather a study of the aspects of use of the environment was interesting. Therefore participants with different skill levels were recruited.

3. *When to evaluate* – As a step in the iterative process of developing the environment, the conceptual model and the following prototype need to be evaluated by users to be able to make adjustments to the existing design. Therefore, feedback from the users at an early stage of the development process is imperative to be able to map out the users’ needs in a design environment as soon as possible.
3.3.1 Evaluation Techniques

I chose to apply three evolution techniques that would help me get the end users’ opinions and observer their performance in the design environment. The techniques I applied were:

- Observing users
- Testing users
- Asking users

The users performed a set of tasks while being filmed in a controlled laboratory setting (observed). While performing three predefined tasks (Appendix E) the participants were requested to verbalize their thoughts (tested). The statements were recorded on tape and video camera. Before and after each session short interviews were conducted to get their opinions of the environment (asked).

**Observing users** The observation of users is the action of looking, listening and recording to mend the gap between what people say they do and what they in fact do (Jordan and Henderson, 1995). Observing the use of an interactive system can offer a great deal of information about what users do, the context in which they do it, how well the technology supports them and what other support is needed (Preece et al., 2002). It can be applied early in the design phase to understand users’ needs, or later in the development phase to test if a prototype meets users’ needs. The latter approach was applied in this study. However, a disadvantage of using this method is time and costs involved (Patel and Davidson, 1995). There may be enormous amount of data produced during the observation, but this may vary dependent on what the observer want to achieve.

A usability study is an approach to observing, which evaluate the ease-of-use and ease-of-learning of an interactive system (Preece et al., 2002). A system or a prototype is being used and tested by users while the observer looks at what are being done. It is often task-oriented testing; a user gets a set of predefined task to perform. There are two methods for usability testing; unobtrusive observation and obtrusive observation (D'Hertefelt, 1999). Unobtrusive observation is a method where the observer refrains from interacting with the users. The observer should avoid from influencing the user as much as possible, by explaining the design or asking questions. With this method one can reveal whether the system is easy to use or not.
Chapter 3: Methods for Design and Evaluation

(D’Hertefelt, 1999). Obtrusive observation on the other hand, is a method where the observer interacts with the user by asking questions, explaining design decisions, or engaging the user in a discussion. With this method issues about the usefulness and acceptance of the system can be revealed (D’Hertefelt, 1999). Neither methods are better than the other, but with the use of both methods the observer will learn a lot about the usefulness and usability of the design. One way is to start out with an unobtrusive observation while doing the set of predefined task and after that, ask questions, explain design decisions and answer the user’s questions (D’Hertefelt, 1999). This is the way the usability study in this evaluation was conducted; starting out with a video recording of every keystroke, mouse click and conversation, and ending up in a conversation with the users that supplemented the observation.

Since observing involves recording both verbal and bodily behaviours which may be difficult to notice and record manually, video recording was used. Furthermore, the observer’s records may often be biased, while video recordings are free of such and can display the actual event (Jordan and Henderson, 1995). Another advantage is the permanence of the observations. The video can be viewed an unlimited number of times, which enable the observer to discover new phenomena, not discovered at the first viewing. It may also be available to other researcher that can extend or disagree with the original analysis (Jordan and Henderson, 1995). However, at some point the video needs to be transformed into words which will involve some loss of information, and may suffer from the bias of human mind. The technology itself, also poses some limitations on the observations. It lacks the full sensory apparatus of humans, e.g. smell, heat and emotions (Jordan and Henderson, 1995). And the decided focus of the camera may influence what is being recorded and miss certain events (Preece et al., 2002). The focus in this observation was on the computer screen to record every movement done in the design environment, and would therefore miss any actions going on outside of the screen. These kinds of decisions may impoverish the recording to some extent. This is important to be aware of when analysing the material (Jordan and Henderson, 1995). Another concern which is important to address is the influence of camera on the people being observed. There has been detected some influence of behaviour, but over time people forget its presence and behave in a normal fashion. The effect wears off quickly, especially if there is no operator behind the camera (Jordan and Henderson, 1995). In this experiment, the camera had no operator, and was placed a couple of metres behind the participants not to be in anybody’s way.
Testing users There are several techniques that can be applied in a usability study, e.g. thinking-aloud protocol, co-discovery learning, question-asking protocol, and performance measurement. However, the think-aloud protocol is the most common, and the technique used in this evaluation (Haak and Jong, 2003). It is a technique to verbalize thoughts, feelings and opinions. This is a quite common process in everyday life; we explain and justify solutions to others. This may require a higher level of cognitive process prior to verbalization than in ordinary communications (Ericsson and Simon, 1984). It is also easier to explain a solution while working on it, than explaining the actions done in the past (Someren et al., 1994). There are two types of this protocol; concurrent think-aloud and retrospective think-aloud. In a concurrent think-aloud situation the participants work and verbalize their thoughts simultaneously. This will give an insight into the participants’ cognitive processes. It may reveal doubts, irritations, surprises and other feelings that may arise during the session (Haak and Jong, 2003). The disadvantage with the use of this method is that the verbalization of thoughts can lead to reactivity (Haak and Jong, 2003), in the way that the participants may work differently as a result of the thinking aloud process. Their performance may either be better or worse. Neither is a desirable outcome, i.e. potential user problems may not be discovered, or false alarms may be generated. In the retrospective approach, the participants first work silently with their tasks and verbalize their thoughts in retrospect. This is a combination of working silently, which resembles a more natural working situation, and then thinking aloud. The disadvantage with use of this method is the problem of remembering the situations that arose during the working phase, and that the participants may invent thoughts which never were present under the experiment (Haak and Jong, 2003). A concurrent think-aloud approach was applied in this thesis to catch the participants’ immediate response. However, the immediate response may not always be as clear and prepared as one should wish. The participants may suddenly pause in a sentence or not complete a word, and easily be disturbed. These actions may be difficult to transcribe later (Someren et al., 1994). However, it is recommended that all sounds, events, and gestures be recorded, because they can have unexpected influence on the participants performance (Someren et al., 1994). The transcription tries to be as faithful as possible to the events happening in each think-aloud session, so that later every aspect of the session could be analysed.

Asking users The third evaluation technique applied was interviewing. There are several types of interviewing techniques; individual, group interviewing, questionnaires, and telephone surveys. They can be structured, semi-structured or unstructured depending on how
rigorously the interviewer sticks to a set of prepared questions (Fontana and Frey, 1994; Preece et al., 2002). I used both structured and unstructured, individual interviewing. In a structured interview, the respondents are asked a set of prepared questions, either with a limited set of alternative answers or open-ended (Fontana and Frey, 1994). This might make it easier to evaluate and compare the respondent. However, categorization of respondents may sometimes limit the field of enquiry. Therefore unstructured interviewing is more appropriate in some cases, and feel more natural, because of resemblance to a conversation (Fontana and Frey, 1994). There are no predefined questions and guidelines, and interesting topics can be followed. However, these interviews may be time consuming and difficult to analyse (Preece et al., 2002).

Before each think-aloud session, a short, structured interview was conducted. The questions were open-ended, and the answers were part of the categorisation of the participants according to programming and Flash experience. The background information of each participant was used to enhance the observer’s ability to analyse the study objects and actions during the experiment. Since the verbalization could be limited at times, supplementary questions were needed. Therefore an unstructured interview was conducted after each session as a cool-down exercise. I had prepared a couple of topics that was to be explored, e.g. what was difficult/easy, and possible improvements. However, the participants were free to speak of what they felt was appropriate. These answers supplemented certain actions taken during the experiment.

3.4 Summery of Methods

I have chosen to employ a qualitative, exploratory approach in this research study. I have focused on conceptual modelling, prototyping, and usability study, which I feel have been the right choice considering the resources and time available.

Creating a conceptual model was for me a way of organizing the theories available, and be able to create a blueprint of the environment that later could be implement. Since the literature available in the area of design environments were quite limited, developing something concrete that users could test was essential to acquire more knowledge. The design of the high-fidelity, horizontal prototyping gave me the opportunity to evaluate the conceptual
model of a design environment. Displaying different features instead of complete functionality was sufficient at this stage of the design process, but should be reconsidered in the redesign process. The prototype was not meant to be evolved further into a full functioning design environment, and therefore possess some limitations.

The usability study showed the aspects of use in the environment, and potential areas for improvement. To get as wide a spectrum of opinions as possible, participants with different backgrounds was recruited. The structured interview conducted before the usability study was part of the categorization of users’ programming skills and Flash experience. While the unstructured interviews conducted after the usability study supplemented the think-aloud process, especially in those cases where the end users’ behaviours were not verbalized or evident to the observer.
Chapter 4: Implementation and Interaction Scenario

The conceptual model consisted of three units as described in section 2.4. However, when turning it into a physical design, certain aspects of the units had to be discarded and others were unexpectedly limited. The most apparent was the reduced computer initiated critiquing component, and the reasoning aspect of the argumentation component. This was either due to design choices made by me which will be discussed in the following chapters or by the tools (Macromedia Flash MX Professional 2004, Macromedia Robohelp X5 and Sothink SWF Quicker 1.7) used in the implementation which will be described in this chapter. The transformation from conceptual model into a prototype may be seen in Figure 4.1:

![Figure 4.1: Transformation of conceptual model into the prototype](image)

A factor that pervades the prototype is the integration into a Flash environment. Since the design environment was directed towards the modification of Flash applications, the term integrated Flash design environment may be used in this thesis. However, in this chapter and the chapters to come I will use the terms design environment or prototype to refer to the integrated Flash design environment.

This chapter will give a thorough description of the design and implementation of the conceptual model into a prototype of a design environment. I will describe the problems encountered in the development process, and the solutions that were chosen to overcome these.

4.1 Development Tools

In both the development and the use of the design environment different software components and tools were applied. A content authoring tool, a scripting language, a SWF editor, a compiler and decomiler, and a help authoring tool were the most important of them. To better
comprehend this division of use and development, the prototype can be divided into two phases as described in Chapter 2; development time and design time. At development time Macromedia Flash MX was used as the main tool. By help of Macromedia Flash the integrating component (Flash application) and logic behind the end-user design activities was implemented. ActionScript was used to glue the integrating component into the rest of the design environment by offering a handle from the Flash application into the environment. At design time, the prototype consisted of a Flash Player, a SWF editor with integrated compiler and decompiler, and a help system.

4.1.1 Macromedia Flash

The integrating component was made up by an application developed in Macromedia Flash. Macromedia Flash is a widespread authoring tool for developing advanced animations, multimedia and applications for the web. 96% of all desktops globally have a Macromedia Flash Player installed (Macromedia, 2005).

Because Flash is designed for the web, Macromedia has placed limitations on what it can do for security reasons. Especially, the use of the sandbox security model may introduce some challenges. Because of this, Flash applications cannot read files from hard disk or communicate with other systems or applications that are not part of the domains it originally came from. The limitations that make the Flash content safe for users to run from the web, makes it somewhat difficult to run in a desktop environment (Chambers, 2002).

One reason why Flash is so widespread in use may be due to its interface. The interface is based on metaphors from the movie industry, and can be divided into three parts:
The timeline area – In the timeline area the content is organized and controlled over time by frames, layers and scenes. It can be compared to an editing room (Larmand, 2005), which consists of three main parts:

- **Frames** – A frame is one moment in time, and the timeline is made up by many frames. A playhead will indicate the current frame displayed at the stage.

- **Layers** – The layers are listed in the right side of timeline area, each with their own timelines. They help to organize and maintain the content. By placing objects in different layers it is possible to do modifications to one layer without affecting objects in another layer. Each layer is transparent, so every layer in the same frame is displayed at the same time. The layer concept is the same as in other graphical applications like PhotoShop, Illustrator, PaintShop etc.

- **Scene** – A scene is a segment of the movie with its own timeline with layers and frames. This allows the user to use to organize the content thematically. For instance, introduction, main content and credits may be in separate scenes.
• *The stage area* – The stage area is where the media content is placed. It provides a preview of how different objects will be displayed in the movie. The stage can be compared to the performance stage in a theatre (Larmand, 2005).

• *The tool panel* – The panel contains tools that let you draw, paint, select and modify the content. This can be seen as the writer's lounge and make-up room (Larmand, 2005).

The animation made in Macromedia Flash displayed in Figure 4.3 consisted of five layers which were called menu, sound, buttons, police and background. This was done to easier maintain the code in the menu layer from the pictures without any interference from any of the other layers. Both the background pictures and buttons were static pictures. The car on the other hand, was made up by many different images to make the illusion of motion. When right-clicking in Windows and Control-click in Macintosh a menu of three different choices would appear. The different menu items (customize image, integrate sound and modify behaviour) were handles into the SWF editor that allowed tailoring.

![Figure 4.3: Animation in Flash Player 7](image)
The menu comes up after right-clicking on an object.
4.1.2 ActionScript

ActionScript 2.0 was used as the scripting language in the development of the prototype, especially as a gluing component between the integrated component and the design environment. It is an object-oriented scripting language, which follows the ECMA-262 specification (ECMA, 1999). ActionScript is part of Macromedia Flash, and adds interactivity to Flash applications and objects like animations, audio, text, and client-side logic. It is not a requirement to use ActionScript in order to use Flash, but to be able to develop a more advanced application and provide user interactivity, ActionScript is recommended.

ActionScript 2.0 is the current version of ActionScript in use. This version was first introduced with Flash 7 (MX 2004). In contrast to ActionScript 1.0 that was not really object-oriented, ActionScript 2.0 added strong typing and object-oriented features to the language. This includes explicit class declarations, inheritance, interfaces, and encapsulation. The language is a high-level and domain-specific programming language (DSL) which means that the commands should be easy to understand because they are domain-specific i.e. movie industry, functions. The DSL hides unnecessary details, and provide enough control on the matters that the users are interested in. There is a restriction in use because of domain specification, i.e. generality is traded for ease-of-use and expressiveness (Mernik et al., 2003). However, by the introduction of ActionScript 2.0 which is object-oriented, the language are getting closer to being a general-purpose language (GPL) that supports classes, subclassing and inheritance.

Below two code examples are shown, taken from the ActionScript Language Reference for Macromedia Flash MX 2004 (Macromedia, 2004a) to show the syntax of the language:

- **gotoAndPlay():**

  ```javascript
  stop();
  myBtn_btn.onRelease = function()
  {
    gotoAndPlay("newFrame");
  };
  myOtherBtn_btn.onRelease = function()
  {
    gotoAndPlay("sceneTwo", 1);
  };
  ```
gotoAndPlay() is a command that sends the playhead to a specific frame, and goes on playing from that frame. In this example, there are two buttons; myBtn_btn and myOtherBtn_btn. When myBtn_btn is released the playhead moves to a frame called newFrame, and when myOtherBtn_btn is released the playhead moves to frame 1 in scene two (sceneTwo) and continues playing.

- getURL():

var listenerObject:Object = new Object();
listenerObject.onLoadInit = function(target_mc:MovieClip) {
    target_mc.onRelease = function() {
        getURL("http://www.macromedia.com/software/Flash/Flashpro/", "_blank");
    }
};
var logo:MovieClipLoader = new MovieClipLoader();
logo.addListener(listenerObject);
logo.loadClip("http://www.macromedia.com/images/shared/product_boxes/159x120/159x120_box_Flashpro.jpg", this.createEmptyMovieClip("macromedia_mc", this.getNextHighestDepth()));

getURL() is a command that loads a document from a specific URL into a window. In this example, a movie clip is loaded and waits until it is clicked. Then an image is loaded by calling getURL() into the browser.

**Customizing Context Menu** To let the end user choose the level of tailoring, a context menu was added. A context menu is a built-in command that let you customize different items in the already existing menu in the Flash player. It is possible to add new items or hide built-in items in the menu except from the “Settings and Debugger” item. Below is the context menu of the prototype with some modifications (for original context menu code, see Appendix A):

var newMenu = new ContextMenu();
newMenu.hideBuiltInItems();

/**EXTENSION***************************************************/
var editCode = new ContextMenuItem("Modify behaviour", extension, true);
/**INTEGRATION***************************************************/
var editSound = new ContextMenuItem("Integrate sound", integrate, true);

/**CUSTOMIZATION*************************************************************************/
var editImage = new ContextMenuItem("Customize image", customization, true);

newMenu.customItems.push(editCode, editSound, editImage);

The menu items: customize, integrate and modify were added as handles into the SWF editor and the rest of the design environment. During development time, three challenges that needed be solved were encountered when implementing the context menu. They will be explained more thoroughly below:

- **FSCommand**
- SWF file vs. EXE file (projector)
- BAT file

**FSCommand** Because of Flash’s sandbox security model, the Flash Player are completely cut off from the outside world. There are however, two exceptions; the functions fscommand() and getURL(). getURL() takes a URL address as a parameter, and “loads a document from a specific URL into a window or passes variables to another application at a defined URL.” (Macromedia, 2004b)

The disadvantage of using getURL() in this prototype, is the need to pass a specific URL address, which contradicts the intension of being able to employ the application on any computer in the future. fscommand() on the other hand “lets the SWF file communicate with either Flash Player or the program hosting Flash Player, such as a web browser” (Macromedia, 2004c). The function takes two parameters; a command and an argument. There are six different commands available:
To launch an external application which was needed in this prototype, the `fscommand()` needs to use the `exec` parameter. The `exec` command is severely crippled because of security reasons. It is no longer possible to pass arguments, and all programs that will be launched need to be in a direct subfolder of the executing Flash file, named `fscommand`. This meant that the SWF editor and help system had to be placed in the `fscommand` folder. This was cumbersome. Instead batch files were used. They called MS-DOS commands to open the two applications. The items in the menu corresponded to three different functions. Each of the functions called `fscommand()` to open the batch files. With some small modifications to the code, it looked like this (for original batch files codes, see Appendix B):

```plaintext
function extension(){
    fscommand ("exec", "Quicker_code.bat");
}

function integrate(){
    fscommand("exec","Quicker_sound.bat");
}

function customization(){
    fscommand("exec","Quicker_movieclip.bat");
}
```

<table>
<thead>
<tr>
<th>Command</th>
<th>Arguments</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allowscale</td>
<td>True or false</td>
<td>Specifying false sets the player so that the animation is always drawn at its original size and never scaled. Specifying true forces the animation to scale to 100% of the player.</td>
</tr>
<tr>
<td>Exec</td>
<td>Path to application</td>
<td>Executes an application from within the projector. In Macromedia Flash MX this application must be in a folder named “fscommand” which is a direct subfolder of the projectors.</td>
</tr>
<tr>
<td>Fullscreen</td>
<td>True or false</td>
<td>Specifying true sets the Flash Projector to full-screen mode. Specifying false returns the player to normal window mode.</td>
</tr>
<tr>
<td>Quit</td>
<td>None</td>
<td>Closes the projector.</td>
</tr>
<tr>
<td>Showmenue</td>
<td>True or false</td>
<td>Specifying true enables the full set of context menu items. Specifying false removes all the context menu items except About Macromedia Flash Player.</td>
</tr>
<tr>
<td>Trapallkeys</td>
<td>True or false</td>
<td>Passes all keystrokes (including ESC and function keys) to the projector.</td>
</tr>
</tbody>
</table>

Table 4.1: Commands and arguments to FSCommand in Macromedia Flash projectors
(Macromedia, 2002a)
SWF file vs. EXE file (projector) Another limitation of `fscommand()` is the inability to employ this command together with SWF files. Instead EXE files or projector has to be used, which is a SWF file wrapped in a standalone Flash Player that can be reproduced outside the browser. Because of security reasons; a SWF file sent across the web should not be allowed to execute programs or files on the client’s local drive (Macromedia, 2002b). A projector let the user be in control of whether to run the file or not.

Using EXE files instead of SWF causes a problem with the SWF editor. SWF Quicker is able to import SWF files, but not EXE files. Some adjustments were made so that the Flash Player played EXE files while the SWF editor imported and exported SWF files.

**BAT file** Since the exec command is not able to pass any arguments, the prototype needed a way to open a specific file (SWF file) within SWF Quicker. Macromedia states:

> “exec is not capable of opening a specific file with an application, just the application itself. One way to open files is to use exec to launch a Windows batch (BAT) file or Macintosh AppleScript file that then opens files in the desired application.”

(Macromedia, 2002a)

Since the prototype was developed in a Windows environment, batch files were called. Three different BAT files were made and placed in the fscommand folder. Each of them called SWF editor along with the SWF file and the corresponding help file. Example of the BAT file called by the `fscommand()`:

```bash
/**The echo command is disabled, i.e. the batch file commands are not displayed on the screen as they are being executed**/
@echo off
start "...\SWFQuicker" "...\Flashfile.swf"
start "...\RoboHelpfile.chm"
```

### 4.1.3 Compilation and Decompilation

Files and applications available for end users are often in an executing format, offering no possibility to do modifications to the source code. However, by decompiling the executing
code (or target program), the source code can be rebuilt. This was the case with the Flash application used in this thesis. To do modifications to the executing Flash application, the running code needed to be decompiled into the source code. After the modifications the source code needed to be compiled back into executing code again. An ActionScript compiler and a decompiler therefore had to be a part of the construction component of the design environment. Since a compiler and decompiler is part of the prototype, and a decompiler basically does the opposite of a compiler, only a description of the logics of a compiler is given below.

Even though, there are thousands of programming languages and compilers, the basic tasks are essentially the same (Aho et al., 1986). A compiler is a program that reads another program written in one language called the source code, and translates it into another language or machine code, often called the target language.

There are two parts in a compilation process; analysis and synthesis. The analysis part breaks the source program into pieces for analysis and creates an intermediate representation. The synthesis part constructs the target program from the intermediate representation. These parts can be divided further into compilation phases as shown in the figure below:
Figure 4.5: Phases of a compiler
(Aho et al., 1986)

The first three phases belong to the analysis part. The lexical analyzer is the first phase of the compiler, and is responsible for the initial reorganization of the top level elements of the program. These elements are: numbers, identifiers, begin and end blocks, statements and program units as functions, procedures, etc. The output is a stream of tokens (Aho et al., 1986). The next phase is the syntax analyzer also called parsing. It groups the tokens into grammatical phrases, and builds hierarchical trees. A syntactic tree makes it easier for the semantic analyzer to detect errors in the operators and operands. The output is an intermediate code that is input for the intermediate code generator phase. The code optimizer improves the intermediate code to a more faster-running machine code. And finally, the last phase generates the target code, that usually consists of machine code or assembly code. The symbol-table manager and error handler are present at each phase. The symbol-table is a data structure that contains a record of each identifier. The different phases enter information about the identifiers into the table. The error handler deals with the errors detected by the different phases. A decompiler translates the target code into source code, and reverses the phases of the compiler.

I wanted the ActionScript compiler and decompiler used in the prototype to be easy-to-use for end users with limited ICT skills. By searching the Internet, few compilers and decompilers
that fitted this criterion were found. One of the search results was Sothink SWF Quicker. It is primarily a SWF editor, but also has integrated ActionScript compiler and decompiler. I therefore chose to apply this program.

4.1.4 Sothink SWF Quicker

Together with the ActionScript compiler and decompiler, the SWF editor (SWF Quicker) made up the construction component of the design environment, and will be referred to as the SWF editor in the remaining chapters. SWF Quicker is primarily an authoring tool, quite similar to Macromedia Flash. Much of the functions found in Macromedia Flash are also found in SWF Quicker which might make it more manageable for end users. The user interface is also inspired by the movie industry metaphors in the same way as Macromedia Flash.

The difference however, between Macromedia Flash and SWF Quicker is the ActionScript decompiler. When a SWF file is imported into the SWF Quicker environment the code is automatically decompiled. The editing capabilities of SWF Quicker was also important, without it, it would have been impossible to change images, sounds etc in a Flash animation.
There are also some drawbacks in choosing this application. SWF Quicker only supports ActionScript 1.0 and not version 2.0 that is integrated into the newer versions of Macromedia Flash. SWF Quicker is able to decompile ActionScript 2.0, but is not yet able to compile it to be a part of the new SWF file. Because the context menu is a function only supported in ActionScript 2.0, SWF Quicker is not able to compile this function. All the other ActionScript functions in the Flash application will execute, except for the context menu that will not appear when a user right-click. When contacting Sothink, they revealed their plans to support ActionScript 2.0 in the future (Appendix G).\(^2\)

Another disadvantage in SWF Quicker is the poor implemented error messaging. It allows no modification, and the messages are often incomprehensible for the end users. This message system is too limiting to function as the critiquing component connecting the SWF editor to the help system. Therefore, a human intervention mechanism had to be implemented as critics. This is explained more thoroughly in section 5.4.2.

### 4.1.5 Macromedia RoboHelp

To develop a design environment, a help system as a part of the feedback component is needed. Help systems has become a vital and integrated part of nearly every user-oriented application. There are several help authoring tool on the marked, designed to different operating systems and applications. To help users with the tailoring tasks a help system was needed, especially a context sensitive help system that supported the users with relevant information about the present task. If for instance, a user needed help to modify the programming code, the help file would display information about ActionScript.

Macromedia Robohelp is a tool to create professional help systems and documentation for desktop and web-based applications. It allows the user to create help systems in different formats, like HTML, Flash, Word, etc.

---

\(^2\)After finishing the experiment, I found out the Sothink now supports ActionsScript 2.0 in SWF Quicker 2.0.
One disadvantage of RoboHelp is that it is impossible to modify the executing file without modifying the source file and translate it into one of the supported formats. This is a drawback since there always are some modifications that need to be done to the help files during the lifetime of an application.

4.2 The Prototype

By the integration of a Flash application, the prototype can be seen as outlined in Figure 4.8. In this prototype, the SWF editor is the construction component, and the help system is argumentation component. Furthermore, researcher’s intervention (which will be discussed in the next chapter) is the critiquing component, and the end-user developer is the designer in the environment.
Figure 4.8: The prototype of the integrated design environment

The directions of the arrows show the way of influence and triggering in the prototype, e.g. the end-user developer does modifications or extensions on the integrated Flash component in the SWF editor, while violations and breakdowns in the editor trigger the researcher’s intervention that alerts the help system. The help system provides suggestions and help that influence the end user to continue or do changes to the processes. This process is evolutionary and terminates only when the end-user developer is satisfied with her modifications.

Within the design environment there are possibilities to tailor a component in gradual steps from direct manipulation to programming. Every object can therefore be tailored according to the three levels of tailorability (Mørch, 1995).

As mentioned in Chapter 3, the design environment can be classified as a horizontal, high-fidelity prototype. This is to show the different features with some path into the functionality of the target system. From every object in the Flash movie the end user can follow three paths of tailoring: customization of objects, integration of sound, or modification of code. The paths start with a menu that appears when an end user right-clicks on an object in the Flash component. This executes the SWF editor and the help system as shown in Figure 4.9:
• **Customization of picture.** At this level the end user had the possibility to customize any object in the application by resizing, moving or changing parameters. These activities were done in the edit and replace editor in the SWF editor.
• *Integration of sound.* At the next level the end user had the possibility to change and add new sound to fit new modifications. These activities were done in the sound editor in the SWF editor.

• *Modification of behaviour.* At the last level the end user had the possibility to modify or add new code by writing ActionScript code. These activities were done in the action editor in the SWF editor.

Within the SWF editor there are several panels, like properties panel, action panel, layout panel, scene panel etc. Activities associated with customization may be found within all these panels, but the overall task is object editing. To make it more comprehensible for the end user the activities within customization was grouped as edit and replace editing, the activities within integration of sound as sound editing, and the activities within modification of behaviour as action editing.

If an end user wanted to change the police car in the Flash application to another type of car, the user could chose “customize police car” in the menu which led to the launch of the construction component in edit and replace editor modus. If the user wanted to do changes to the implementation code on the other hand, the action editor in the construction component would open. Because of limitations for modification in the COTS (Commercial off-the-shelf) itself, some adjustments to the original prototype design had to be made. Instead of getting straight into the edit and replace editor, the user had to find the panels associated with edit and replace editing in the SWF editor themselves.

### 4.3 Interaction Scenario

To understand how the design environment can be used by end users, an interaction scenario of an end user doing design activities is provided below:

In order to really understand the Doppler Effect, and use the Flash application in another setting, an end user decides to do some modifications. Instead of a police car driving by, the end user wants to replace it with an ambulance. By right mouse clicking on the executing application a menu with three options appears. The options are customize image, integrate
sound and modify behaviour. Since the end user wants to replace the police car, customize image is chosen in the menu. This action executes the SWF editor and the help system (see Figure 4.9). The SWF editor imports the Flash application, along with decompiling the ActionScript code. The help system that appears contains help files relevant to the task at hand, namely replacing images. The application is now ready to be edited by the end user.

The end user is a novice when it comes to Flash editing, so she starts to read the help file named “Customize movie clip”. The help file suggests the use of the functions “replace symbol…” and “edit symbol…” located in the property panel. The end user therefore clicks on “edit symbol…” and enters into an edit modus, and then chooses “replace symbol…” A library with different design units appears. By searching the library, the end user finds an image of an ambulance that can be used. She clicks on the ambulance, and the police car is replaced by the ambulance. However, the movie needs to be set back to normal modus if the new replacement is going to be active. The end user searchers and tries different buttons to get out of edit modus, without any luck. At this moment, an intervention is activated that direct the end user to the file “edit symbol…” within the help system. A suggestion on how to get out of editing modus is followed, which results in getting back to the original movie, but now with an ambulance instead of a police car. The end user feels satisfied with the result and chooses to export the movie and compile the ActionScript into a new Flash application (SWF file).

However after testing the new version of the application, the end user finds the sound of the movie not quite right since there is a police siren with the ambulance. The siren needs to be changed. The end user therefore right mouse click in this new Flash application and chooses “integrate sound” in the menu, and continue further on the process of integrating a sound file in pursuit of tailoring the application.

4.4 Summery of Implementation

While turning the conceptual model into a physical design, different questions about the tools arose. Macromedia Flash places some limitations and restrictions on their tool that makes it more difficult to freely modify and customize the tool and the Flash Player, after the users’ needs. Their sandbox security model makes it hard for the Flash Player to communicate with
files and applications outside its own domain. Because of an increasingly hostile web environment, it is impossible to use the `fscommand()` and SWF files together outside the application's original domain.

The SWF editor as the construction component also contained limitations that affected the prototype, which have to be taken into consideration when evaluating the conceptual model. It did not support ActionScript 2.0 which is standard scripting language with all recent Macromedia Flash products. The problem arises when the script is to be compiled into a SWF file. The SWF editor’s messaging system was too poor to function as the critiquing component. It had no possibilities for modification or connection to the help system. Therefore a human initiated critiquing component had to be implemented into the prototype. A better solution to argumentation component would have been to build it on the basis of collaborative authoring environments. This would have given the end users the possibility to insert, delete, or modify help information. The help system used was static in the way that every bit of new information had to be recompiled into a new help file.

The implementation of the prototype has revealed features with the conceptual model that need to be reconsidered in a later redesign. Nevertheless, the prototype was not intended as a pilot for a future design environment. Therefore, the problems and limitations encountered are excellent cases for learning how to improve the design environment.
Chapter 5: Experiment and Analysis

The result of the evaluation phase is analysed in this chapter, while the conclusions outlined in the next. The chapter starts with a description of experiment and the participants involved. Then the observations are presented, along with an analysis of the collected data.

Analysis is the process of making sense out of collected data. The data or phenomenon is broken down into smaller pieces to understand it more fully (Colapietro, 1993; Creswell, 2003). The goal is to find patterns, themes and categories of the data material (Patel and Davidson, 1995). The observation done during the usability study and discoveries of the implementation of the prototype are broken down to show common tendencies or irregularities during the experiment. The pieces are then put together into a context to be discussed and compared with results from former research studies.

The excerpts of the transcripts in this chapter are translated into English. The relevant parts of the original transcription are in Norwegian (Appendix F). The transcription, tasks and description of the study are provided in the appendix.

5.1 Experiment

A vital part of this thesis is the usability study done on the design environment. The study was conducted during two days in InterMedia’s usability lab. Each session was recorded on audio and video. By use of image and sound recordings, both verbalization and visual hand movements could be analyzed.

5.1.1 Experimental Set-up

The design environment, as mentioned in Chapter 4, consisted of a Flash player integrated into a design environment that consisted of an SWF editor, an ActionScript compiler and decompiler and a help system. All the incorporated components are products intended as standalone applications, but in the context of this thesis they were combined to create a Flash integrated design environment for the participants.
The integrated Flash application simulating the Doppler Effect was the object of modification (Figure 4.2). The reason for creating a simulation of the Doppler Effect was to show a possible approach for demonstrating a physical phenomenon, and creating educational application units by use of Flash. The Doppler Effect is taught in high school physics, and may therefore be quite recognizable. This simulation consisted of a background image of a city, a road with a moving police car, and a siren with an increasing or decreasing pitch according to the relative position from the viewer. A start button sat the car (and the siren) in motion until a stop button was pressed. When a participant right mouse clicked on an object, e.g. city, car, road, button, or sky, in the Flash application a context menu appeared with three choices. By choosing the first menu item (customize image) the participant could change, modify or add objects to the animation, by choosing the second item (integrate sound) the participants could integrate different sound files to the animation, and by choosing the last item (modify behaviour) the participant could extend or modify existing program code.

**Experimental design** Each participant was first given a consent form to sign, and then a description of the computational environment (Appendix C). To avoid spending too much time getting to know the complete functionality of the SWF editor, a short introduction to general Flash environments was given verbally. Then the task description (Appendix E) was handed out. The first task was divided into two subtasks with increasing level of difficulty. The participants should first replace the background image with another and then replace the moving object. The different images and objects were available in the SWF editor’s library. The participants could if they wanted, put self-selected images and objects into the library. This feature was not chosen. I chose to put them in myself to avoid spending participants’ time on finding appropriate images. In the second task the participants had to change the existing sound file of a police siren into an ambulance siren. The sound files of the sirens were also pre-imported into the library. The third and last task was a programming task; the participants had to replace a bit of the code with an ActionScript command sequence. This would lead to a change in the behaviour of the car; from driving endlessly, in a loop to only driving across the screen once. Context sensitive help, which I will refer to as the help system throughout this chapter, was provided at every step of the problem solving.

The four independent usability tests were conducted in a sound-isolated room. The layout of the room is shown (Figure 5.1):
Figure 5.1: Layout of the experimental setting

The design environment ran on the laptop, but showed on a bigger monitor and keyboard for the participants’ convenience (Figure 5.1). All modifications was first saved on the laptop and then transferred to a server belonging to the university. I was present as an observer throughout the entire session to make sure that the computer equipment, application software and camera worked, and that the participants verbalized their thoughts. Even though I was supposed to be a silent observer, I had to remind the participants of the purpose of “think aloud”. Furthermore, I wrote down interesting user activities, and let them serve as “index” into the video recording protocol. This facilitated later search for interesting events. The video camera was focused on the screen to capture all mouse movements. These recordings together with sound recordings and observations are the basis of my analysis.

5.1.2 Participants

I have earlier in this thesis (Chapter 3) justified my choice of participants. In this section I will try to group them according to their background and skills, to be better capable of making judgement about their performance in the study.

Five participants with different backgrounds, gender and ICT skills were asked to take part in the experiment, of which four volunteered. The participants’ identities are made anonymous,
and I will therefore refer to them as participant #1 to #4. The participants were grouped according to the following parameters:

1. General programming knowledge:
   - Beginner – no knowledge of programming languages
   - Intermediary – knowledge of one or two programming languages
   - Advanced – knowledge of more than two programming languages

2. Flash experience:
   - None – never developed applications in Flash
   - Intermediary – developed one application in Flash
   - Extensive – developed several applications in Flash

The classification below (Table 5.1) is based on answers given by the participants when asked questions (Appendix D) about their background before the actual usability study:

<table>
<thead>
<tr>
<th>Participant#</th>
<th>Programming knowledge</th>
<th>Flash experience</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Intermediary</td>
<td>None</td>
</tr>
<tr>
<td>2</td>
<td>Advanced</td>
<td>Intermediary</td>
</tr>
<tr>
<td>3</td>
<td>Beginner</td>
<td>None</td>
</tr>
<tr>
<td>4</td>
<td>Advanced</td>
<td>None</td>
</tr>
</tbody>
</table>

Table 5.1: Classification of participants

Three out of four participants had some programming experience, mainly from university courses and projects. All the participants, had seen or used Flash on the web, but it was only participant #2 that had tried to develop a Flash application. Participant #4 had studied and tested some functions of the tool, but never developed and could therefore only be classified as having no Flash experience before the usability study started. All the participants were offered an introduction of the Flash environment where I explained concepts like timeline, scenes, layers and frames.

Each of the participants was given one and a half hour at their disposal to finish the tasks. However, there was no time pressure; none of the participants knew how much time they had
used or how much time they had remaining. Nevertheless, none of the participants needed more than one hour to finish the tasks. Therefore, time is not taken into consideration when the results are analyzed; it only reflects the skill level of the participants.

5.2 Observations

The observations are divided into two sections dependent on the factors; tailorability in the design environment and use of the design environment. Observations done within these two groups are distributed as follows:

- **Tailoring:**
  - Modification by direct manipulation
  - Programming by extension and modification

- **Use:**
  - Trial and error
  - Reflection support
  - Breakdowns

Each subsection will start out with an introduction. Then segments of empirical data are presented to illustrate my findings. Finally, the observations are interpreted and discussed theoretically by comparison with former research studies. The signs and symbols used in the transcription of the empirical data are adjusted from Silverman’s original list of simplified transcription symbols (Silverman, 2003):
Table 5.2: Transcription symbol
(Silverman, 2003)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>R:</td>
<td>Researcher talks</td>
</tr>
<tr>
<td>()</td>
<td>Unintended or untranscribable utterance</td>
</tr>
<tr>
<td>.hhhh</td>
<td>Audible inhale, number of h’s indicates length</td>
</tr>
<tr>
<td>hhhh</td>
<td>Audible exhale, number of h’s indicates length</td>
</tr>
<tr>
<td>()</td>
<td>Indicates a short silence or pause</td>
</tr>
<tr>
<td>(4)</td>
<td>Pause or silence measured in seconds</td>
</tr>
<tr>
<td>CAPITAL</td>
<td>Words in caps indicate a louder voice relative to the adjacent talk</td>
</tr>
<tr>
<td>(())</td>
<td>Author’s descriptions rather than transcriptions</td>
</tr>
<tr>
<td>Oka::y</td>
<td>The “::” indicates a lengthening of the proceeding vowel sound</td>
</tr>
<tr>
<td>Exactly</td>
<td>Word said in a slower, emphatic fashion</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Indicates an action made by the participant</td>
</tr>
<tr>
<td>?</td>
<td>Indicates a rising intonation</td>
</tr>
<tr>
<td>.</td>
<td>Indicates a stopping fall in tone</td>
</tr>
</tbody>
</table>

5.3 Tailorability in the Integrated Flash Environment

The order of the tasks reflected the gradual increase in complexity starting with direct manipulation and ending up in programming. By making the tasks increasingly more advanced, the participants were supposed to feel a sense of achievement after each task and a motivation to start on the next one (Newell and Simon, 1972). Despite this, the setting may have caused some of the participants to feel uneasy: “I become nervous in this setting. I feel that I should have understood more and worked faster, and as a result becomes more stressed.” (Participant #4). A perceived pressure to perform and the artificial setting may have influenced the participants’ ability to solve the tasks. This fact must therefore be taken into consideration when analysing the performance of the participants.
5.3.1 Modification by Direct Manipulation

Direct manipulation was involved in two of the tasks in the usability study, either by replacing images or sound. This section will therefore be divided into two subsections. The first will be about direct manipulation of images and the other about direct manipulation of sound.

Direct manipulation of images Replace the background image was the first subtask, while the second subtask was about replacing the image of a moving object which involves replacing the image in every frame of a particular layer.

Task 1: Change the background picture of the Flash application from a picture of a city to a rural picture. Then change the police car into an ambulance.

The participants started on Task 1 by searching for the image to be replaced by going through the layers in the SWF editor. All images had been placed in layers, when the Flash application had been imported into the editor. The layers were annotated by numbers, which gave no indication of its content, so the participants spent some time on finding the right layer:

Participant #3:
1: <clicks on different object of the movie in the SWF editor>
2: I’m only clicking around to find out which layer it <an image> belongs to <the layer of an image becomes activated when the image is clicked>
3: the houses <layer number> are 12?
4: <drags the image of the house out of the editor screen of the SWF editor, this will activate the layer>
5: YES

Even though, the layer was found, it did not give the participants any indication of further moves. It therefore led the participant to use the help system. The help system suggested an order of commands to follow for replacement of images. The “edit symbol…” and “replace symbol…” links in the property panel should be used. Nearly every participant struggled to find the location of this panel. Even after reading the help system that explicitly stated its location, the property panel was hard to find. The location at the bottom of the screen and the fact that it changed appearance when used in different settings may be a reason, i.e. sound
modus vs. editing modus. Below is an image that shows the location of the property panel in editing modus:

![Figure 5.2: Location of the property panel in the environment](image)

Other approaches to replace the image were therefore tried. One of these was the use of right mouse click, which was a success. The two links was found in the right mouse click menu, and the property panel could be ignored for now. Participant #2 tried out this approach:

1: <double clicks on the city image and enter into editing modus>
2: here is the city at least
3: <right-clicks on the city and chooses replace symbol. A replace window appears>
4: if we replace this <refer to the city image> with something different
5: this seems correct (4) and was it symbol 5? <chooses symbol 5 from a list in the replace window>

The second subtask seemed easier than the first one, because of some repetition of commands. However, a peculiar combination of functions in the SWF editor made the task a bit harder than the first. To replace a given image in one frame, it was sufficient to use the “replace symbol…” function. However to replace an image in every frame of the SWF file to imitate a moving object, an edit modus had to be entered before it was possible to replace the image. This was explained in the task description, but few understood its meaning before they had tried and failed a couple of times. The difference in skill level was apparent in the way the
tasks were solved. Three of the participants with the most ICT experience tried to transfer knowledge of functions from other applications to this environment. This could be observed when they searched for layers, which is a common way of representing information in graphical applications and environment. It was also apparent in the use of right mouse clicks on objects to search for functions. Even though assistance from the help system was provided, it was only participant #3 that really followed the instructions given to replace the images.

In this task, modifications were done to the Flash application, but without use of any programming. As a result the changes were limited, but at the same time more manageable for less experienced participants, because only meta-level modifications are involved (Pryor and Bastán, 1999). The activities performed were deleting, finding and replacing images by modifying parameters of already existing application components. Direct manipulation involved in customization, is one of the activities that separates end-user tailorability from end-user development (Mørch, 1995; Costabile et al., 2003). Customization involves no real modification, only selection among predefined configurations (EUD-net, 2003). It is therefore not classifies as an end-user development activity, but as an end-user tailoring activity (Mørch, 1995).

**Direct manipulation of sound** Task 2 was quite similar to the first task, only in this task a sound file was to be replaced. The Flash application had originally a moving police car with a police siren. The police car was replaced with an ambulance in Task 1, and therefore the sirens had to be replaced in Task 2.

Task 2: *Change the sound of the application from a police siren to an ambulance siren.*

As with the previous task, the participants found the task somewhat difficult to perform. The location of the property panel in sound modus (see Figure 5.2 above) was not easy to find. The participants had an especially hard time locating the property panel in the sound mode, as seen in this example involving participant #2:

1: I’m thinking that there should be a place somewhere to change the sound
2: It is here somewhere if I just can find it
3: <looks around in the SWF editor> cannot say that I see any place to change the sound.
Participant #2 resorted to the SWF editor’s help files in this case. When finally finding the property panel, he noted: “has sound been there (referring to the property panel) all the time and I haven’t seen it. It is almost embarrassing.” The sound layer however, was much easier to find than the correct image layer, because of the graphic equalizer placed within it. Even thought, the sound layer was visible, it gave no indication of the whereabouts of the property panel.

In the first task some of the participants used the right mouse button to change images and disregarded the property panel. This was not possible in the second task. The property panel was the only place in the editor to replace sound files. They all managed it in the end, either by having to resort to the help files, or the help system.

To avoid the use of the property panel in this task, the participants had to import an external sound file and modify it to the conditions of the application. By letting the participants integrate an external sound component, and fit this into the application the end user activity performed would have been classified as integration (Mørch, 1995). Integration is supposed to surpass customization by allowing new functionality to be added to the application, without writing any code. Since the replacement of sound file did not involve any advanced functionality like macro recording, script recording or advanced copy and paste functionality, it cannot be classified as integration merely base on the activity performed by the end user. However, the sound file was pre-imported and fitted into the rest of the application in advance to save time on searching for appropriate sound files, and fitting it according to length and pitch. Therefore the activities of pre-importing and modifying the file, along with the end user activity performed, Task 2 may be classified as integration.

5.3.2 Programming by Extension and Modification

The last task was a programming task. In this task the participants were supposed to make the ambulance stop after it had driven across the screen once. This was to be done by programming in ActionScript, which is a domain-specific language (DSL) that uses commands and functions from the application domain of movie making (Moock, 2004). The contradiction between ease-of-use and expressiveness of general-purpose languages (GPL) and DSL was relevant during this task. However, because of varying programming background among the participants, the use of a DSL seemed reasonable in this task.
Task 3: *Change the behaviour of the car from driving endlessly to stopping after driving by the screen once.*

In a Flash application, different layers and frames may have different code segments. In this application there were mainly two different layers that contained different codes. These were the code for the menu (Appendix A) and the code for the moving object. During development time these codes were placed in a menu layer and a moving object layer, respectively. However, when the Flash application was imported into the SWF editor all the code was placed within the same layer, but in different frames. The participants therefore had to find a particular frame within the action layer in the SWF editor, which included the command: `gotoAndPlay(2);`. This is a loop command that orders the application to go back to the beginning of the movie and play it again. To stop this behaviour, the participants had to replace `gotoAndPlay(2);` with a command that terminated the action. The different commands they could use: `stop();` or `gotoAndStop();`.

Finding the right frame containing the command `gotoAndPlay(2);` proved to be difficult for the participants. However, hints in the help system indicated which frame to look for. When the right frame was located, the code became visible in a separate window (the action panel). As opposed to finding the property panel, locating the action panel where the programming to take place therefore seemed easier. Figure 5.3 below shows the action editor with functions like syntax check, and command list, etc.

![Figure 5.3: Action editor in Sothink SWF Quicker](image)
Participants #1, #2 and #4 had some programming experience, with a minimum of one programming language learned in university courses. Participant #3 however, did not have any programming experience. It was therefore especially interesting to see how she solved the last task:

1: can I for instance simply write "stopp" ((Norwegian))? Shall I try this?
2: <removes the existing code and writes “stopp”>
3: do not think this will work (.) I can try preview and see if anything happens
4: <clicks on preview button, the car stops after driving by the screen once. When trying for a second time, an error message appears alerting of an error in the code>

Since the car is supposed to stop, a command that indicates this action was tried. She wrote stop in Norwegian (stopp). This was neither the right ActionScript syntax nor within the right language (English), which of course was corrected later. This showed that the intuitive suggestion was not far from the right answer; stop();. Instead of guessing, participant #2 remembered from earlier development in Flash that gotoAndStop(); could be used for this. There were no restrictions on which command to use, but stop(); was maybe the most intuitive of the alternatives, and the rest of the participants used this. The difference between the other participants and participant #3 was the fact that even though they did not use the right command at the first trial, the syntax was correct, which probably indicates some previous programming experience.

Previous experience with programming languages could be observed for instance in participant #1, which was categorised as having intermediary programming knowledge. He understood the gotoAndPlay(2); command by looking at it: “It seems like it is a loop function. If I’m going to guess”. Participant #1 started to test commands by using the member list button, and suggested the stop(); right away:
However, to get the correct syntax in ActionScript a parentheses and semicolons are needed, but it is sufficient with only semicolons. By using the member list button as shown in Figure 5.4, parentheses and semicolon are not inserted. This had to be added by the participants. The transcripts below illustrate this when participant #1 was trying to insert the correct command:

1: I’m going to insert a stop instead <removes the previous command and insert stop by use of the member list>
2: no (.) <Tries the check button which validates the code. It returns an error message>
3: I can try this then <insert a semicolon after the stop command>
4: <validates the code again, with the help of the check button which returns the message “no error found”>

Participant #1 used the member list and check button actively after reading about them in the help system. However, the need for a semicolon was not mentioned anywhere. Therefore the suggestion to insert a semicolon may have resulted from previous experience with other languages. A general syntax rule of programming languages is often that a command ends with a semicolon (e.g. Java, C++).

Despite of the programming experience, participants felt some anxiety when the idea of programming was presented. Participant #3: “I panicked when I saw “modify behaviour” (an item in the Flash application’s context menu), because I realised that I had to code”. The thought of doing modifications on the base-level of the application can be daunting for users. The use of DSLs are supposed to bridge some of the gap between meta-level and base-level by using terms from on the task domain, and coding on the meta-level, but making
modifications to the base-level, (Mørch et al., 2004). Using commands from the application domain, like `stop()` make it easier for the novice programmer. This corresponds to Nardi’s findings of spreadsheet languages. She suggested that the primitives of languages should be task-specific such as `sum()` and `average()`, and corresponding to the application domain (Nardi, 1993). The participants used the help system as soon as they realised that coding would be involved. However, after they realised that the task was manageable, they all managed to complete it. Participant #3 even said: “I think I could have been good at this, if I only had spent more time on it”.

With this task as with Task 1, the separation between end-user development and end-user tailoring becomes apparent. According to Mørch (Mørch, 1995) only activities that involve programming by use of GPL can be classified as an activity at the tailoring level of extension, while EUD-net (EUD-net, 2003) follows Costabile (Costabile et al., 2003) in defining scripting as an end-user development activity. However, since there are no rules of which programming language to use in EUT (Mørch, 1995), programming by extension using object-oriented DSL, will therefore in this thesis be classified as an extension task (Mørch, 1995). The use of DSL might have become restricting for the more advanced programming participants if they had been asked to code once more. A GLP may have been preferred for advanced programming support. In this task, however, the use of DSL was sufficient.

5.4 Use of the Design Environment

Aspects of use of the integrated Flash design environment were studied, along with tailorability of the environment. Design environments have been developed and studied several times before (Fischer et al., 1989a), but an environment where standalone component is integrated into the environment, is to my knowledge not yet studied. The Flash Player was used as the integrating component, and the entire environment was tested for its usability and usefulness. The advantage of using Flash is that it is commonly used to add interactivity to web pages, thus most users have a basic understanding of its functionality.

A design environment consists of three components: construction, critiquing and argumentation (Fischer et al., 1998). Standalone applications combined into one environment, was used to model this. The connection between the components was realised by using a
scripting language (ActionScript) and DOS commands to make the transition as seamless as possible. This was to some extent successful as one of the participants stated after entering the editing environment: “…this (SWF Quicker) is probably a standard editing tool for Flash files” (participant #1). In fact, SWF Quicker is one of several SWF editing tool.

5.4.1 Trial and Error

Throughout the entire study the method of trial and error was evident among the participants. Instead of using the help system or the help files integrated into the SWF editor, the participants opted for trial and error instead. The help mechanisms would only be used in extraordinary situations where no other obvious alternatives were left. Participant #1 explained this as follows:

1: I’m really a trial and error person.
2: I turn to help files after a lot of trying (.) when I am using a program
3: in this case I tried to use the help files sooner (.) but that was a bit confusing

This comment was made after the participant experienced some problems solving Task 2, and the help system had to be used. During or after the experiment, all the participants mentioned that they preferred this method when using a new application. Participant #2 also commented on the use of the trial and error method: “I usually only use help files when I am completely stuck. I simply feel that I learn more by trying first, and then if that doesn’t work… maybe then”

This tendency could be observed throughout the entire experiment, often as an approach to familiarizing themselves with the application. Participant #2 and #4 even found an alternative and much easier way to solve Task 1 by this method. It gave them the opportunity to get to know the different functions of the application in their own way, and a sense of achievement when the method led to a success. It was quite “safe” to experiment in this way and few users experienced failure using this approach. Small steps were taken by constantly testing out different approaches to improve the next.

The trial and error method is explained by Schön (1987) as a step-by-step process of reflection. According to this view, after a success or failure a period of reflection follows.
This is used to plan the next move to improve the participant’s next action. A surprise, either pleasant or unpleasant, causes the situation to talk back to the users; this will influence their next step. The reflection is not visible for an observer, or even for the user himself. It can however, be observed as an on-the-spot experiment (Schön, 1987). In the design environment, a participant may try actions used in similar applications, like right mouse clicking on objects, or try to program with intuitive, domain-specific commands as done in the transcript in section 5.3.2. Another sign of reflection is use of help system(s). This is similar to Fischer’s suggestion of using an argumentation component to simulate the situation’s backtalk (Fischer et al., 1991). The need for argumentation occurs when the users are stuck in a problematic situation and need to think about what to do next (Fischer et al., 1989b). The participants could become reflective practitioners (Schön, 1983) by using their newly found insights to improve the next step.

5.4.2 Reflection Support

In the process of reflection described above, a support component can improve the users’ ability to reflect. The SWF editor provided integrated help files, while the design environment provided the participants with an argumentation component in the form of context sensitive help system. Some confusion could be observed with the different help systems. The context sensitive help system seemed difficult to notice, and some of the participants therefore used the SWF editor’s integrated help files instead. This help files needed to be searched by keyword or free-text to find information. The confusion is demonstrated in the transcripts below, where participant #1 requests help while solving Task 1:

1: are help files associated with the F1 button? (.) I’m a trial and error person
2: <uses SWF editor’s integrated help files, without any success>
3: R: Is this <SWF editor’s help files> the only help files available
4: <clicks on the help system that is situated on Window’s taskbar. Closes them after a while >
5: are there more than one set of help files? <clicks on the help files one more time>
6: do you have your own help files?
7: I’m supposed to do this in Sothink <the SWF editor> (4) now I’m a bit confused
Participant #2 however, never used the help system. He replied when asked about this after the session was finished: “no (had not seen them)… do see now that there are several help files”. The other participants used the help system at least once during the problem solving, but not more than four times. Even though, participant #1, #3 and #4 knew of its existence, they had to be reminded of its use several times.

According to Fischer (1998) actions in the construction component should trigger actions in the argumentation using a “backtalk” mechanism such as a critiquing component. The help system in this design environment was triggered by three events:

1. Choosing an item in the context menu of the integrated Flash component.
2. Requiring help to resolve breakdowns
3. Researcher’s interventions

Except for participant #2, the participants became aware of the help system when they chose an item in the context menu. However, they rarely read them right away. The help system was also activated by the participants themselves when breakdown occurred in situations where no other possible on-the-spot experiments or trials could be thought of. Researcher’s intervention was a reminder initiated by the researcher. This will be described in the next section.

Conventional help files already existed in the SWF editor. Here search for help could be done by use of keywords or free-text. However, a common problem is that users may not be aware of their own inadequacy to solve a task, and do not know that relevant information is available through the conventional help files. If however, the users are aware of its existence, they may not know which keyword to use in a search, and therefore has trouble retrieving useful information (Fischer et al., 1989a). This was observed when one of the participants tried to use the SWF editor’s help files, but did not succeed because she did not know what to search for. The help system tries to mend this by help adapted to the current task available at all times. It supports the realisation of reflection (Schön, 1983; Fischer et al., 2002) in the construction process, especially by arguing and reasoning about a design (Fischer et al., 1989a). In the Janus project (Fischer et al., 1989a) there was a tighter integration between the construction component and the argumentation component by the use of a critiquing component. In the integrated Flash design environment the critiquing component had to be replaced by the use of researcher’s interventions. This had to be done because the SWF editor
lacked a satisfying messaging system or the possibility for modifications. The researcher’s intervention will be described in the section below.

5.4.3 Breakdowns

Breakdowns during a construction process may, if proper support is given, enable the user to reframe problems and find solutions (Fischer, 1993). It may lead to new understanding and knowledge. During the experiment the participants experienced two types of breakdown; breakdowns because of violations of design principles, and breakdowns because of no new suggestions (trials). Both breakdowns led to the use of either the SWF editor’s help files or the help system. This section however, is going to look closer on two different triggers for use of help mechanisms; computer initiated intervention and researcher’s intervention.

**Computer initiated intervention** Violations in design or programming principles, will if the application has integrated computer initiated interventions, result in a trigger to execute error messages. The SWF editor harboured such a function. However, error messages can be quite cryptic at times, especially for inexperienced users. When solving Task 3 which involved some ActionScript programming, participant #3 experienced a computer initiated intervention. The intervention occurred because of an error in the syntax, and the cryptic error message that appeared in the SWF editor read: “ERROR: Scene 1, Layer 1, Frame 75, Line 2: Unexpected EOF found while looking for input.” Participant #3 with no programming knowledge read this message, but could not understand it. There was no reference to any help files or hints on how to solve this error.

A breakdown is not a negative situation to be avoided, it is a situation of non-obviousness (Winograd and Flores, 1986). It makes the user stop and reflect upon the situation, to find the next step. In this case however, the participant understood that there was something incorrect in the code, but not what was wrong. She had no possibility to get any situational backtalk (Schön, 1983) because the SWF editor did not provide any help files to follow the computer initiated intervention. In a design environment breakdowns should trigger a help mechanism. In Janus a critiquing component was integrated with the construction and argumentation components (Fischer et al., 1989a). There are two agents involved in a critiquing system; a computer and a user working together. The user’s role is to generate and modify solutions while the computer’s role is to analyze the solutions, and produce critics that the user can
utilize in the next step towards the solution. In this design environment however, researcher’s interventions were used to accomplish the same.

**Researcher’s interventions** Since the design environment did not have any satisfying computer initiated interventions; a human intervention mechanism had to replace it. In this thesis this mechanism is called *researcher’s intervention*. When breakdowns occurred, either by error messages from the SWF editor or by lack of any new trial and error generated suggestions, an intervention form the researcher was needed to make the participants aware of the help system. To be able to help participant #3 to complete the third task, a researcher’s intervention was applied:

1:  R: if you try to read the help files again (.) there is supposed to be a place where all commands `<ActionScript>` are listed <referring to the member list button in the action editor>
2:  <Finds the member list button>
3:  Ah it suggests stop (.) I wrote it in Norwegian (.) no wonder why it didn’t work

Since the computer initiated intervention only provided the participant with an error message without any help on how to comprehend the message or ways to solve the programming task, I had to intervene. The intervention reminded the participants of the help system and indicated an ActionScript function that could be of interest. Researcher’s interventions were also applied to other participants and other tasks. The situation above, however, differs some from the rest because it demonstrates the shortcomings in the computer initiated interventions provided by the SWF editor, and the need for a supporting help mechanism in the design environment, at a higher (more user oriented) level of abstraction.

The actions taken in the SWF editor does alert an intervention, but the limited intervention mechanism did not provide any opportunity for follow-up reflection (Schön, 1987; Fischer et al., 1991). In the Janus project (Fischer et al., 1989a) breakdowns in the construction component alerted the critiquing component that gave the user an opportunity to reason about the design because of the argumentation component. The argumentation was given in the form of a design rationale (Fischer et al., 1989a). The critiquing component as implemented in Janus, was not without limitations. Both user goals (tacit and explicit) may be difficult to pick up by a computer. Because of limited information available in the environment, the support
may not always fit the situation. The support may therefore sometimes be a hinder for the design process instead of an enrichment (Fischer, 1993). Human interventions however, address this to some extent. They have the possibility to get a holistic view of the situation and tailor the help according to user needs, as well as raise it to a higher (user oriented) level of abstraction. This is still something computers are not very good at yet.
Chapter 6: Summary and Conclusions

To complete the evaluation process and to move forward into the redesign phase, conclusions must be drawn. In this Master’s thesis I have studied the conceptual model of a design environment, implemented a high-fidelity prototype and conducted a usability study. The intention was to contribute to the development of an end-user development (EUD) environment that supports end users in design activities, focusing on modification of Flash applications. Besides studying the realisation of the conceptual model, the possibilities of end-user tailoring activities (EUT) within the design environment were evaluated.

The purpose of this chapter is to summarize the study and draw tentative conclusions about its usefulness. The conclusions together with directions for further study may contribute to the evolution of the conceptual model and design environment into a better system.

6.1 Analysing the Conceptual Model

The conceptual model was influenced by two other research studies done within the field of EUD environments. Janus (Fischer et al., 1989a) showed that a domain-oriented design environment (DODE) could help the end users in obtaining relevant information to solve problems and accomplish tasks of kitchen design. KitchenDesign (Mørch, 1996) showed how to evolve generic applications into a DODE by end-user tailoring. Using these as background, I could focus on the tailorability of Flash applications within an EUD environment. The transformation and evaluation of the conceptual model into a physical design showed that the model presented in this thesis is highly feasible, and that placing it within a Flash environment is technically possible. The following summarizes and concludes about the observations made in connection with the different parts of the conceptual model and their implementation into the prototype.

6.1.1 The Prototype of an EUD Environment

The conceptual model was made up by three parts; the design environment with the three components of construction, critics and argumentation, the tailoring activities of customization, integration and extension, and learning by designing and being critiqued. Each
of these will be studied below to show to what extent and how they were implemented into the prototype.

**Design environment** At the heart of the conceptual model, were the theories and ideas of an end-user design environment. A theoretical framework was provided by the former research studies. The functions of the three components (construction, argumentation and critics) have been described by Fischer (Fischer et al., 1989a; Fischer et al., 1989b; Fischer et al., 1998), but no specific descriptions on how to implement them have been given. Therefore, the three components were implemented without specific guidelines using a SWF editor, researcher’s interventions, and a HTML based help system.

The construction component was implemented using a SWF editor with integrated ActionScript compiler and decompiler. It provided the end users with the opportunity of developing and modifying Flash applications. Developing in the SWF editor is based on moviemaking metaphors like *timeline*, *stage* and *editing tools*. This is provided to conceptually separate the different developing phases of the application from each other. However, the study of the prototype revealed that the habits of the end user do not correspond with these divisions. Well-known functions like right mouse clicking on the operating object in the stage area were for instance preferred over doing modification in a separate editing window. The SWF editor was not consistent in allowing the user to do direct manipulation by standard mouse clicking operations, which caused some confusion. A tighter integration between the stage area and editing tools may therefore be preferred. Because of a WYSIWYG (What You See Is What You Get) interface, the timeline metaphor was however accepted, i.e. all participants intuitively understood its meaning during the usability study. The end user could move backwards and forwards in the Flash movie by moving the playhead.

Another feature of the SWF editor was the ready-to-use design units of images, sounds and code modules. These were placed in different libraries in the environment, and were quite useful when solving the tailoring tasks of the usability study. However, the visibility of the libraries in the SWF editor could have been improved, especially the code modules, which the end users experienced some difficulty in locating. Therefore, visible and static library windows, displaying design units, may have simplified the design activities.
The critiquing component was implemented in two different ways; computer initiated interventions and researcher’s interventions. The SWF editor provided some computer initiated interventions that became active when violations to design and programming principles were broken. However, they proved to be incomprehensible and insufficient, especially for the novice end users. Therefore researcher’s interventions were added. Appropriate critic feedback at the right time and right level of abstraction was applied, making the information better adjusted to the user and the situation. However, the limitations of these two forms of interventions were mainly caused by poor integration with the help system. The SWF editor did not allow modifications to the computer initiated interventions, thus no automated connection could be implemented. Instead the help system was connected to specific (predefined) tailoring activities to simulate integration. This supported the end users in coping with breakdowns related to the tailoring activity, but not with unrelated (tailoring) breakdowns in the environment, such as changing colours or text, etc. During unrelated breakdowns the end user’s ability to reflect upon the situation and the knowledge creation process may have been reduced because of little feedback. However, measuring the learning process involved is difficult and was not within the scope of this thesis, and therefore the possible outcome of breakdowns can only be taken as speculations. In the future a more refined computer initiated critiquing component integrated with the help system, should be developed. To be able to implement this, another construction component than the SWF editor, should be used that permits such integration.

A help system was implemented as the argumentation component. The help system provided the end users with help to resolve problems that arose during design activities. The use was variable; some end users utilized it more often than others. The reasons for this mixed reception may be: 1) The end users had different levels of experience in developing Flash applications and programming skills. 2) In some cases the integration between the critics and the help system was insufficient. In these cases the support had to be accessed on the end users’ own initiative. 3) The location of the help system; placed in a separate window of the SWF editor, was hard to locate. The first reason gives no raise for concern, and the second was dealt with in the previous paragraph (human interventions). The third reason, however, may be resolved by for instance giving the help files a more visual appearance within the construction area, e.g. a permanent window.
Even though the help system provided help for certain critical situations, it provided only minor arguments and reasoning for applying the suggestions. The lack of reasoning around the suggested solutions, may have crippled the possibility for reflection-in-action (Schön, 1983). The end users may find it difficult to change behaviour when no arguments for the proposed solutions are given. This may only be speculations. However, implementing argumentation and reasoning into the help files should be done in future versions of the environment. To improve this component even further, it may be turned into a collaborative environment, e.g. Wiki or blog. This would provide the end users with the opportunity to edit the argumentation, and collaborate with each other during design time. They would also learn for each other by providing multiple perspectives on a solution. By for instance implementing the argumentation as a semi-structured blog, end users may provide advice and discuss design decisions collaboratively, e.g. using the comment functionality of the blog.

**End-user design activities** The design environment supported mainly three activities; 1) direct manipulation by editing images, 2) integration of sound, and 3) programming by extension and modification. There were no restrictions on the kind of design activities that were possible to perform according to the conceptual model or within environment per se, but I wanted to focus on tailoring activities. The SWF editor however, placed some restrictions on the type of activities that could be performed, supporting mainly direct manipulation, visual programming and object-oriented scripting. These activities can be placed within both EUD and EUT, and therefore shows that certain end-user development and tailoring tasks can be performed within the design environment. The study however only focused on the EUT activities of customization, integration and extension (Mørch, 1998). Section 6.3 will look closer at the implementation of the EUT activities.

**Learning by designing and being critiqued** As mentioned before it is difficult to measure the learning and knowledge creation involved in performing the tailoring tasks in the design environment. This is a process that takes place within each end user, which even pedagogical experts may find difficult to measure. I will however, claim that the process of learning by designing and being critiqued was present in the environment through the model of the three implemented components (construction, argumentation and critics). Breakdowns during the tailoring activities activated the researcher’s or the computer initiated interventions that pointed to suggestions for alternative solutions in the argumentation component (help system). A process of reflection upon the situation would then be initiated by the end user.
Reflecting upon the situation may then initiate knowledge creation process. However, since the integration between the interventions (researcher’s and computer initiated) and the argumentation was not technically implemented, and that the argumentation component was not complete, some of the prerequisites for supporting reflection-in-action may not have been there. Despite this, during the experiment there were some indications of learning: 1) a trial and error method was used, which according to Schön, is a step-by-step process of reflection that gradually improves the user’s next action (Schön, 1983), and 2) a notable improvement in the time to solve the problems because of increasing familiarity with the Flash environment and functions, even though there was an gradual increase in complexity. Therefore it may seem that the design environment displayed some qualities supporting learning by designing and being critiqued.

6.2 Integrating the Environment into a Flash Domain

The technological affordances and constraints offered by Flash placed some limitations on the development and possible activities of the environment. The Flash environment was not a direct part of the conceptual model, but pervaded every aspect of the implementation of the prototype. The SWF editing tool was discussed above, so in this section the focus is on Macromedia Flash as the development tool of the prototype.

The development of the design environment showed that executing a Flash application (SWF file) within or outside a Flash environment implies certain challenges. Executing a Flash application within the Macromedia environment poses no limitations on the functionality. However, outside the environment new rules apply. Restrictions are placed during runtime in pursuit of protecting against malicious intent, with special focus on security risks in uncontrolled environments like the Internet. At the same time as serving to protect, these safeguards place restriction on the actual use of the Flash application. Three factors that limited the development of the design environment were identified:

1. The Sandbox security model – Macromedia Flash adhere to a sandbox security model, described in Chapter 4. This is to make it safe to run Flash applications from the Web, but makes it difficult to run in a desktop environment. It has resulted in the following restrictions:
• To execute a file or program from within a Flash application, the file or program needs to be placed in a direct subfolder to the running Flash application. However, this can be avoided by calling on a BAT file that executes the file or program.

• Distributing a Flash application as a SWF file outside of the original domain are also limited. The intention is to let the receiver of the file be in control of executing or disabling it. Therefore an EXE file of the Flash application must be used, which was the case in this study.

2. **Decompiling** – Macromedia Flash has no tools for decompiling ActionScript programmed applications, and cannot provide support for editing an existing SWF file or EXE file. Therefore an independent SWF editor with an integrated ActionScript decompiler was used as the combined construction component of the design environment. It may seem unlikely that Macromedia Flash will support decompilation in the future because of their proprietary nature. However, several software applications offer this feature such as Sothink SWF Quicker and Flash Decompiler, which may lead to a reconsideration of their policy.

3. **ActionScript 2.0** – ActionScript 2.0 is an object-oriented scripting language that is tightly connected to Macromedia Flash. It was not supported by the decompilator of the SWF editor used in the design environment (only ActionScript 1.0). This caused a problem when using the context menu as a handle into the design environment, since it can only be implemented in version 2.0. Therefore the context menu could not be decompiled in the SWF editor. This will only be a temporary problem with this version of the SWF editor.\(^3\)

### 6.3 Evaluating the End User Developing Environment

The usability study tested three tailoring activities supported by the design environment. *Customization* was performed by replacing images, *integration* was performed by integrating a new sound file, and *extension* was performed by programming in ActionScript. All

---

\(^3\) In Sothink SWF Quicker 2.0, ActionScript 2.0 is supported.
participants managed to solve the three tasks they were assigned, which show that the design environment supported these activities.

**Direct manipulation** The first two tasks of customization and integration involved direct manipulation by the end users. Objects were replaced using direct manipulation techniques. The first task followed a step-by-step replacement process that ended up producing a new image. Customization is classified as an EUT activity (Mørch, 1995; Costabile et al., 2003). However, there are disagreements whether customization also can be classified as an EUD activity or not. This is on the basis that no real modifications to the application take place during customization only changes among predefined configurations. However, in this thesis direct manipulation is seen as an end-user development activity when performed in the context of a design environment as can be seen in Table 2.1.

The second task showed that integration was possible in the design environment, even though the actual usability task only involved direct manipulation. The end user was to replace a sound file with another. I had reduced this task to involve only replacement to save time during the usability study. The sound file was pre-imported and fitted into the rest of the application in advance. The integration of a new sound object into the application, was accomplished by end users and me in collaboration, and may therefore be classified as joint integration activity. The end user could, however, have done it themselves, but much time would have been spent on searching for an appropriate file, and thereafter fitting it according to length and pitch. I felt that this was unnecessary to prove that the environment supported integration.

**Programming** The environment supported both visual programming and object-oriented scripting. The last task showed the programming possibilities in the design environment. This task was to program by modification and extension, using a scripting language. This showed the extension possibilities of the environment. According to EUT, extension should involve programming by use of a general-purpose language (GPL) and not a domain-specific language (DSL) as in this case, but this is not a fixed rule (Mørch, 1995). Since ActionScript 2.0 is class-based object oriented, and therefore close to being a GPL with classes, subclassing and inheritance, the task should be seen as extension.


6.4 Limitations

To maintain a certain focus, restrictions had to be made on what to include in the thesis. I will therefore in this section, mention some limitations that had the greatest impact on this thesis.

To evaluate the conceptual model, prototyping was applied as a design technique. When developing a prototype of a complex environment, limitations are inevitable. Limitations imposed by the choice between developing features contra functionality had to be made. At this stage of the development process displaying the possible features was preferred. I therefore chose to develop a horizontal prototype with the exceptions of three implemented functions. The aim was to study aspects of use, and tailorability of the environment. When studying aspects of use, different features needed to be displayed, while when studying tailorability some functionality needed to be present. Replacing of images, integration of sound and programming were the three functions implemented. However, during the implementation unexpected limitations caused by the tools used, had to be dealt with. These were mostly security measures placed on the tool, and workarounds had to be made.

Other limitations experienced that may have influenced the usability study was the laboratory setting and the use of camera. Conducting a test within a laboratory setting that is supposed to show normal use, is a contradiction in terms. However, I was only interested in usability and not particularly how it may function in the end users’ work environment. The camera may have prohibited the participants in their think-aloud process, but studies show that after a while the camera will be forgotten and the participants will act as normal (Jordan and Henderson, 1995).

The sample size of the usability study is another factor worth mentioning. A sample of four participants may be criticised as a too small. However, according to the formula of Nielsen and Landauer (1993) four participants will find 75% of the usability problems in a prototype. This I find sufficient.

6.5 Directions for Further Work

This study has shown that design environments are an excellent technology in promoting end-user development. However, to support design as a knowledge creating process the three
components (construction, argumentation and critics) needs to be completely implemented. Furthermore, by letting the end user tailor Flash applications at different levels of complexity the end user can gradually improve their mastery of programming. I am therefore convinced that a design environment will benefit the design process and the end-user development community.

There are six directions that I feel further development should be directed and evaluated. The result of these will be quite interesting, and a condition for further improvements of a Flash integrated design environment. The first four directions concern the specific design environment presented in this thesis and the last two address design environments in general. They are as follows:

1. More robust solutions need to be developed to resolve the issues placed on Flash applications by the sandbox security model.

2. Computer initiated intervention need to be implemented and integrated with the help system, and brought to the same level of abstraction as researcher’s interventions.

3. The help system must include support for argumentation and reasoning, along with the existing support.

4. The argumentation component could be implemented as a web-based collaboration environment, allowing the end users to be part of the design discussions (e.g. Wiki or blog).

5. Learning by creating involved in using a design environment should be further explored and assessed.

6. More application design activities need to be identified to see if the design environment has the possibility to support them.

I hope that the results from this thesis may contribute to adjustments and further development of the environment, which would be an excellent focus for another Master’s thesis.
Bibliography


Hakkarainen, K., T. Palonen, S. Paavola and E. Lehtinen (2004). *Communities of Network Expertise: Professional and Educational Perspectives*, Elsevier.


Appendix

Appendix A: Context Menu (ActionScript 2.0)

```actionscript
var newMenu = new ContextMenu();
newMenu.hideBuiltInItems();

/**EXTENSION**************************************************************************/
var editCode = new ContextMenuItem("Modify behaviour", extension, true);

/**INTEGRATION**************************************************************************/
var editSound = new ContextMenuItem("Integrate sound", integrate, true);

var dummy;
newMenu.customItems.push(editCode, editSound, dummy);

for (i in this) {
    this[i].menu = newMenu;
}

newMenu.onSelect = menuHandler;

/**FUNCTIONS**************************************************************************/
function extension(){
    fscommand("exec", "Quicker_code.bat");
}

function integrate(){
    fscommand("exec", "Quicker_sound.bat");
}

function customizeText(){
    fscommand("exec", "Quicker_text.bat");
}

function customizeButton(){
    fscommand("exec", "Quicker_button.bat");
}

function customizeMovieclip(){
    fscommand("exec", "Quicker_movieclip.bat");
}

/**Event Handler**************************************************************************/
function menuHandler(obj:Object, menu:ContextMenu){
    newMenu.customItems.pop();
    var itemName;

    if(obj instanceof MovieClip){
        itemName = new ContextMenuItem("Customize "+ obj._name,
        customizeMovieclip, true,true);
    }
    if(obj instanceof Button){
        itemName = new ContextMenuItem("Customize "+ obj._name,
        customizeButton, true,true);
    }
```
if(obj instanceof TextField){
    itemName = new ContextMenuItem("Customize " + obj._name, customizeText, true, true);
}
newMenu.customItems.push(itemName);
Appendix B: BAT Files

Task 1:
@echo off
cd "C:\Programfiler\SourceTec\Sothink SWF Quicker"
start SWFQuicker "C:\Documents and Settings\Kjersti\Skrivebord\hoytenkningseksperiment\Kandidat2\dopplertest.swf"
start ""
"C:\hovedfag\flashhjelp\SWFeditor_movieclip\!SSL!\Microsoft_HTML_Help\SWFeditor_movieclip.chm"

Task 2:
@echo off
cd "C:\Programfiler\SourceTec\Sothink SWF Quicker"
start SWFQuicker "C:\Documents and Settings\Kjersti\Skrivebord\hoytenkningseksperiment\Kandidat2\oppgave1.swf"
start "" "C:\hovedfag\flashhjelp\Sound\!SSL!\Microsoft_HTML_Help\Sound.chm"

Task 3:
@echo off
cd "C:\Programfiler\SourceTec\Sothink SWF Quicker"
start SWFQuicker "C:\Documents and Settings\Kjersti\Skrivebord\hoytenkningseksperiment\Kandidat2\oppgave2.swf"
start ""
"C:\hovedfag\flashhjelp\SWFeditor_code\!SSL!\Microsoft_HTML_Help\SWFeditor_code.chm"
Appendix C: Description of Usability Study


Denne applikasjonen er en prototyp, dvs. at all funksjonalitet ikke vil være til stede. Derfor er det viktig at du holder deg til de oppgavene jeg har satt opp. Skulle du på noen tidspunkt ikke komme videre pga uklar oppgavetekst eller uforståelig hjelpefile, eller mangel på funksjonalitet kan du bare å ta kontakt med meg. Jeg vil være til stedet under hele forsøket.


Det er viktig at du tester ut Flash animasjonen før du setter i gang med oppgavene, slik at du blir kjent med dens funksjonalitet. Hvis du ikke har vært borte i Fash før, og ikke vet hvordan en slik animasjon er bygget opp, kan jeg gi en liten innføring før du begynner.
Appendix D: Warm-up Questions to Usability Study

Alder:

Stilling/utdanning:

IKT kompetanse:

Hvordan benytter du deg av IKT både privat og jobb-/utdanningssammenheng?

Har du brukt eller benyttet deg av Macromedia Flash eller lignende verktøy privat eller i jobb-/utdanningssammenheng? Hvis ja, forklar hvordan det benyttes.

Har du programmeringserfaring?
Hvis ja, hvilke programmeringsspråk?
Appendix E: Tailoring Tasks of Usability study

1) Du skal først forandre bakgrunnsbildet til Flash applikasjonen fra by til et landlig bilde. Dette gjøres ved at du høyreklikker i applikasjonen og velger ”Customize [picture]”.

   a) Bytt ut by bildet (city) med et landlig bilde. Det landlige bildet ligger allerede klart i Sothink under navnet symbol 5. Bildet må gjøres større slik at det har samme størrelse som det forrige.

2) Etter det skal du skifte ut politibilen med en ambulanse.

   a) Bytt ut politibilen med en ambulanse. Her er det viktig å benytte seg av ”edit” funksjonen før ”replace”.¹

Lagre forandringene ved å klikke på ”Export Movie” i Sothink og kall filen oppgave1.swf (husk å lagre filen i mappen med dit kandidatsnr). Lukk Sothink og hjelpefilen, slik at du bare står tilbake med den opprinnelige applikasjonen.

3) Du skal nå forandre lyden fra politisirene til ambulansesirene. Dette gjøres ved å høyrekikke i animasjonen og velge ”Integrate sound”

   a) Bytt ut politisirene til ambulansesirene.

Lagre forandringene ved å klikke på ”Export Movie” i Sothink og kall filen oppgave2.swf (husk å lagre filen i mappen med dit kandidatsnr). Lukk Sothink og hjelpefilen, slik at du bare står tilbake med den opprinnelige applikasjonen.

4) Du skal nå forandre atferden til bilen. Dette gjøres ved å høyrekikke på et sted i animasjonen og velge ”Modify behaviour”. Etter å ha trykket på ”start” knappen vil bilen kjøre uendelig til noen trykker på ”stopp”.

¹ For å lage en animasjon er det noen ganger behov for å sette inn samme bilde flere ganger i samme layer. For at et nytt bilde skal gjelde for hele animasjonen må vi benytte edit (som editerer for hele animasjonen og ikke bare framen) før replace.
a) Du skal nå få ambulansen til å stoppe etter å ha kjørt forbi skjermen en gang. Dette gjøres ved å legge inn en kommando i action layer’s siste frame.

Lagre forandringene ved å klikke på ”Export Movie” i Sothink og kall filen oppgave3.swf (husk å lagre filen i mappen med dit kandidatsnr). Lukk Sothink og hjelpfilen, og spill av oppgave3.swf.
Appendix F: Transcripts – Original

Task 1a:

Participant #1:

1:  <velger ”customize city” i context menyen, kommer inn i SWF editor>
2:  <klikker på forskjellige steder i editoren> må jeg finne insert layer eller noe sånt?
3:  <Finner ut at by bildet (city) befinner seg på layer12>
4:  <klikker på høyre musetast> hvis jeg inserter en layer over der <layer12> og så (4) <ler>
5:  kan jeg spørre deg om hjelp nå, eller?
6:  (.) for å få lagt inn et nytt bilde (2) var litt sånn
7:  R: mitt forslag er å lese hjelpefiler
8:  er der hjelpefilene på F1 knappen? (. ) jeg er litt sånn prøv og feil jeg da
9:  <benytter seg av SWF editorens hjelpefiler, uten hell>
10: R: Finnes det flere hjelpefiler eller (. ) er det bare de til Sothink
11: <klikker på de hjelpefilene som befinner seg på Windows oppgavelinje. Lukker dem etter en stund>
12: er det flere hjelpefiler? <klikker på hjelpefilene igjen>
13: har du egne hjelpefiler?
14: jeg skal jo gjøre dette i Sothink <SWF editor> (4) Jeg ble litt forvirra nå
15: R: når du høyreklikker <i Flash Playeren> (. ) kommer det opp en hjelpemelding som passer til oppgaven
16: <Foreslår å starte fra nytt igjen. Ser da at det kommer opp hjelpefiler. Leser hjelpefilen som forklarer om kommandoen ”replace symbol…”>
17: prøver å finne replace symbol…
18: <finner det ikke i property panel, heller ikke når det trykkes på høyre musetast.
Klikker på layer12, men da dukker ikke linken opp i property panel>
19: <Må henvise han til hjelpefilene igjen, finner da ut at han må finne property panel. Etter litt leting rundt på skjermen finner han det>
20: <Trykker på ”edit symbol…” (instrukser fra hjelpefile for å bytte bilde). Trykker på undo for å komme tilbake til opprinnelig bilde> Synes ikke dette var så lett
21: <Resonnerer> jeg velger city’en <bildet som skal byttes ut i SWF editoren>
22: <klikker på øyesymbolet (deaktiverer) i layer12, dette gjør at by bildet forsvinner> ok, så ser jeg at hele byen forsvinner
23: på replace symbol… i hjelpefila står det at det skal først trykkes på edit symbol (2) da blir jeg forvirra, for hvorfor kan jeg ikke bare trykke replace symbol…?

24: <Klikker på replace symbol…, får opp replace-vinduet, bytter bildet, og gjør det større. Klikker så på øyesymbolet (deaktiverer) i layer12 igjen og ser at det nye bildet forsvinner og kommer tilbake pga øyesymbolet (aktiveres) klikkes på igjen> Det må jo være greit

Participant #2:
1: <velger ”customize city” i context menyen, kommer inn i SWF editor>
2: <klikker seg gjennom lagene> Bare ser gjennom lagene (.) og ser hva som ligger rundt omkring
3: det første jeg tenker er at det ikke står symbol 5 <det nye navnet til by bildet i SWF editoren, denne informasjonen var gitt i oppgaveteksten> her noen sted
4: <klikker på høyre musetast på objektet air og velger ”convert symbol” fra menyen, convert vinduet kommer opp. Konvertere symbolet air til symbol 5>
5: <en feilmelding kommer til syne>
6: det så ikke riktig ut
7: <Dobbel klikker på city bildet, kommer inn i edit modus>
8: her har vi i hvertfall funnet byen
9: <Høyreklikker på byen og velger replace symbol. Replace vinduet kommer opp>
10: hvis vi replacer den <referer til by bildet> bytter den ut med noe annet
11: Det virket naturlig (4) og var det symbol5? <velger symbol 5 fra en liste i replace vinduet>
12: <Prøver så å få det til å ha samme størrelse som det forrige i edit vinduet. Først ved å bruke en kommando som han har hentet fra Flash. Det fungerer ikke her. Så prøver han å dra det til en viss størrelse. Gir opp>
13: <leser oppgaveteksten igjen> samme størrelse som det forrige
14: vet ikke helt hvordan jeg skal gjøre da.
15: <Undo’er seg helt tilbake til by bildet for å se på størrelsen, prøver å replace bildet igjen og prøver med det blotte øyet å få det like stort.> ja (.) sier oss fornøyd med det foreløpig i hvert fall

Participant #3:
1: <velger ”customize city” i context menyen, kommer inn i SWF editor>
Appendix

2: <klikker på forskjellige steder i bildet i SWF editoren>
3: nå klikker jeg litt forskjellig bare for å prøve å finne hva som er layer <til bildet> <layeret blir aktivert når man klikker på bildet>
4: er husene <layer nummer> 12?
5: <drar bildet av husene bort fra editor vinduet i SWF editoren, dette gjør at layeret blir aktivert>
6: YES
7: finnes det bare en delete funksjon her eller? (. ) er det litt dristig når jeg skal gjøre om landskapet?
8: R: prøv
9: <trykker på delete, bildet forsvinner>
10: men da har jeg plutselig ikke noe å gå ut ifra da
11: <Gjør oppmerksom på hjelpefilene>
12: <Leser disse> er ikke helt sikker på hva et symbol <et bilde kalles symbol i SWF editoren> er?
13: <forklarer at symbol er et bilde>
14: skal jeg bytte city <bildet> med (2) replace kanskje <klikker på replace symbol… linken>
15: <bytter ut bildet og tilpasser størrelsen>

Participant #4:
1: <velger ”customize air” i context menyen, kommer inn i SWF editor>
2: <Leter etter symbol 5, finner ikke det på siden. Leser hjelpefila>
3: må ikke tegne deg selv? (4) men hvor ligger det hen? <leter i hjelpe fila>
4: <Må forklare at hjelpe fil vinduet ikke er editor vinduet, som er der for å leses man kan ikke lage noe i dette hjelpe vinduet>
5: <Høyreklikker på bildet i SWF editoren og velger replace symbol… i menyen. Velger symbol 5>
6: <Prover å forstørre bildet. Leter etter muligheter for dette i menyen til høyre musetast. Finner ingen kommandoer som kan brukes>
7: eller kan man gjøre symbolet større sånn? <drar i bildet>
8: <Bildet bare roteres, leter i menyen til høyre musetast igjen>
9: <Tilslutt må jeg gi hint for at å få forstørret bildet. Bildet forstørres>
Task 1b:

Participant #1:

1: Jeg er egentlig et sånn prøv og feil menneske, for å si det sånn (.)
2: for min del når jeg bruker et program så pleier jeg å ty til hjelpfilene etter at jeg har prøvd mye selv
3: det jeg prøvde å gjøre nå (.) var å se på hjelpfilene litt tidligere, og det synes jeg var litt forvirrende
4: <velger ”intergrate sound” i context menyen, kommer inn i SWF editor>
5: her så det veldig ut som et lydspor <peker mot lyd layeret>
6: <aktiverer hjelpesystemet> får lese her da
7: <klikker på lyd layeret etter å ha lest litt>
8: det var den <lyd layeret> jeg tippa ja (2) <leser videre>
9: R: velg frame i stedet for layer
10: <gjør det. Velger så symbol 1 i stedet for symbol 2 i sound panel>
11: men for å høre på den <nye lyden> (3)
12: nå har jeg bare forandret symbol <lydfila>
13: <letter rundt på skjermen> er det noe sånn play-knapp her
14: jeg får ikke noe lyd når jeg skroller <flytter timeline pekeren fram og tilbake>
15: er det noe lyd på her forresten? <tester ut lyden på den opprinnelige animasjonen> ja der kom den <lyden>
16: R: les litt lengre ned på hjelpesila
17: <klikker på preview knappen og tester ut animasjonen> ja (.) det hørtes i hvert fall ut som en (5)
18: <lager og lukker>

Participant #3:

1: vet ikke helt om jeg får til disse bevegelsene <sikter til bevegelsen av politibilen i Flash filmen>
2: jeg har jo ikke noe politibil i bildet mitt <Klikker på forskjellige frames i filen, da kommer politibilen til syne>
3: <Klikker på replace symbol…, replace viduet kommer opp. Leser oppgaveteksten igjen, der står det at det er viktig å klikke på edit symbol… før replace symbol… Lukker replace vinduet og klikker på edit symbol… linken>
4:  <bilen kommer så i edit modus> skal jeg tegne bilen <ambulansen> nå? Jeg vet ikke helt hvor jeg skal finne ambulansen
5:  <Klikker på replace symbol… og bytter politibilen med ambulansen>
6:  <er usikker hvordan komme ut av edit modus>
7:  R: hva om du bruker hjelpetil
8:  jeg bruker jo ikke (4) <klikker på hjelpetil>
9:  jeg er jo sånn som bruker uten <hjelpetiler> jeg
10: <leser hjelpetil og finner ut at ”scene 1” skal klikkes på for å komme ut av edit
    modus>

Task 2:
Participant #2:
1:  <velger ”integrate sound” i context menyen, kommer inn i SWF editor>
2:  <klikker på lyd layeren> dette så vel ut som lyd.
3:  <Høyreklikker på lyd layeret> tja (3)
4:  <Klikker så på en frame i dette lyd layeret> ”
5:  tenker vel at her burde det ett eller annet sted være et lurt sted å forandre på lyden
6:  det er det vel kanskje også hvis jeg bare finner det
7:  <ser seg rundt i SWF editoren> kan ikke si jeg ser noen lurt sted å forandre på lyden
6:  R: kanskje du finner det ut ved å bruke en hjelpetil
7:  (.) er det da vi skal finne hjelpetiler ja
8:  <Søker i SWF Quicker’s helpefiler etter ”sound”, leser informasjonen>
9:  aha (.) <bytter lydfilen til politisirenen med ambulansesirenen> har det stått sound der
    <referer til property panel> hele tida? (.) også har jeg ikke sett det
10: det er jo nesten flaut
11: <Lagrer og lukker Sothink>
12: <trykker på start på den opprinnelige animasjonen>

Task 3:
Participant #1:
1:  <velger ”modify behaviour” i context menyen, kommer inn i SWF editor>
2:  <flytter på timeline pekeren>
3: Jeg bare scroll’a til sluttens her <siste frame> for å se når den <timeline peker> er her ute
4: aha, gotoAndPlay 2 <referer til kommandoen i actionvinduet i siste frame>
5: R: hva tror du den gjør?
6: det virker som det er den loop funksjonen (1) hvis jeg skal gjette
7: synes det er litt rart (.) fordi det ser ut som hele actionscriptet er delt
8: så hvis jeg (.) <merker kommandoen og letter gjennom memberlist, klikker på kommandoen stop>
9: <stop kommandoen legger seg bak den opprinnelige kommandoen (gotoAndPlay(2))>
10: nei (.) den la til stopp <fjerner stop() kommandoen>
11: <utfører en syntaks sjekk, ved å klikke på check knappen>
12: sånn (3) <fjerner gotoAndPlay (2) kommandoen> skal legge inn en stopp <stop()> i stedet <letter i membelist, legger til stop>
13: <utfører en syntaks sjekk, får feilmelding. Legger til en “;” sjekker koden, ingen feilmelding>
14: <trykker på preview - alt ser bra ut. Lagrer og lukker>

Participant #3:
1: <velger ”modify behaviour” i context menyen, kommer inn i SWF editor>
2: <leser hjelpefilene, finner action layeret i filmen, og letser seg fram til siste frame i layeret der koden skal byttes ut>
3: R: Dette skal være et veldig enkelt språk
4: så jeg kan bare skrive stopp for eksempel? skal jeg prøve det?
5: <fjerner koden som er der og skriver stopp>
6: tror ikke det går helt bra da (.) så kan jeg prøve preview og så kan jeg sjekke ut om noe skjedde
7: <klikker på preview, bilen stopper å ha kjørt forbi skjermen en gang. Prøver preview en gang til, får opp feilmelding som sier at koden inneholdt feil>
8: R: hvis du leser hjelpefilene igjen (.) er det et sted der forskjellige kommandoer <ActionScript> er listet opp <hinter til member list knappen i action editoren>
9: <Finner member list knappen>
10: å ja den foreslår stop (.) jeg skrev det på norsk (.) det er ikke rart at den ikke virket
11: sånn <legger inn kommandoen i koden>
12: <Prøver preview, applikasjonen vil ikke kjøre>
13: <Forklarer at man må bruke ( )<parantes> etter stop-kommandoen, da virker alt som det skal>
Appendix G: E-Mail correspondence with Sothink

Dear Kjersti Sæther,

Thanks for your mail and interest in Sothink SWF Quicker.

Yes, the current version of Quicker just support to show AS 2.0.
We are planning to let it support AS 2.0 in future release.

======================================
If you have any suggestion or comment to our service work, please feel free to contact us via Email.
Your feedback will help us to better understand your needs and improve our service to you.

We appreciate your co-operation!

Sincerely,

Gracie Wong
Customer Service
SourceTec Software Co., LTD
Web: http://www.sothink.com
Email: support@sothink.com

From: Kjersti Sæther
Sent: Friday, April 08, 2005 4:40 PM
To: support@sothink.com
Subject: Sothink SWF Quicker

Hi,

I’m a master student in informatics at the University of Oslo. I studying authoring tools as a part of my master thesis and therefore got interested in your SWF editor. I tried it out and it was very good, but I need an SWF editor that supports actionscript 2.0. Since Sothing SWF Quicker 1.6 does not support actionscript 2.0 I therefore wondered, if you have developed an SWF editor that supported actionscript 2.0 or if you have any plan for doing this?

Yours sincerely,
Kjersti Saether