Prototyping an Intuitive Graphical User Interface for a Presentation Program

Peder Ydalus

Master Thesis
Institute of Informatics

UNIVERSITY OF OSLO
August 15 2006
Abstract

The purpose of this thesis is to provide a study of an implementation of a graphical user interface (GUI) in a presentation program intended for non-expert users. The type of program is chosen because it’s no secret that massive marketing has made Microsoft PowerPoint push all competition off the market, leaving PowerPoint as the only and obvious (but not necessarily the best) choice. There might very well be a void to fill where lack of alternatives makes PowerPoint the only option. For this study, a completely new presentation program will be developed. The program will present an interface with fairly new components for users of varying expertise and the study will attempt to judge how well these elements fit these users, and if possible refine the interface. The output format will be a slideshow generated as Macromedia Flash vector graphics, which can be viewed in most common browsers with the appropriate plug-in, or in a standalone player. In this document the words slideshow and presentation are used interchangeably.

The title of the thesis hints that one of the purposes of the development of this user interface is to make it intuitive. But what really is intuitive? In short, a paradigm for a well designed user interface is that the program behaves exactly the same way the user expects it to. Nevertheless, users are different and both perception and intuition are to some degree concepts related to the individual user. This project will attempt to explore how to make a user interface the general user intuitively understand and don’t need much time to learn. It’s a project goal to create a user interface with an easy learning curve and an environment the users feel comfortable working in. The term “user friendly” will be explored in detail in chapter 7.

As for the program, it has been developed using Microsoft Visual Studio 2002. This development tool takes the full advantage of the .Net platform which in turn simplifies work that previously had to be made with complex Windows API calls. C# was chosen as the programming language.

The name of the application is Switch.
Acknowledgements

Working on this project has been a very giving and challenging process. Challenging is meant in a good way because the knowledge acquired in the process has been equally big. I’m grateful that the Institute of Informatics and my two supervisors accepted that I could conduct this research and would at the same time like to take this opportunity to thank the people that have contributed to this project.

First of all I would like to thank my supervisors Bjørn Faugli and Anders Mørch. They have provided me with productive ideas and feedback throughout the process. I appreciate the support and motivation they have offered me and the final evaluations of the thesis have been invaluable.

Furthermore I’d like to thank the helpful staff at IFI Drift. Without their help and equipment on very short notice, setting up the usability laboratory and conducting the usability tests would not have been possible.

Moreover I’d like to thank Ole André Solbakken for helping me with an evaluation of psychological aspects of the Switch user interface. His feedback was very enlightening.

Finally I’d like to thank those family, friends, colleagues, friends of friends, friends of colleagues and fellow students who have helped me in the prototyping process and the usability testing. Without their help one can only guess what the interface would have looked like today.

Last, but not least, I would like to thank my friends and family for believing in me and supporting me.

Oslo August 15, 2006
Peder Ydalus
# Contents

1. **INTRODUCTION** .......................................................... 1
   1.1 **WHAT IS A USER INTERFACE?** .............................. 2
   1.2 **WHAT IS A COMPUTER INTERFACE?** .................. 3
   1.3 **COMPUTERS AND USER INTERFACES** ................... 4
   1.4 **A BRIEF HISTORY OF USER INTERFACES** ............. 5
   1.5 **A BRIEF HISTORY OF POWERPOINT** .................... 9

2. **RESEARCH PROBLEM** ................................................. 11
   2.1 **THE CORE RESEARCH PROBLEM** ......................... 13
   2.2 **LIMITATIONS** ................................................... 18
   2.3 **MOTIVATIONS** .................................................. 20

3. **RESEARCH DESIGN AND RESEARCH METHODS** ............ 21
   3.1 **QUALITATIVE METHODS** ...................................... 22
   3.2 **QUANTITATIVE METHODS** .................................... 23
   3.3 **A FOUR PHASE INTERFACE DESIGN PROCESS** ...... 24
   3.4 **RESEARCH TEAM MEMBERS** ................................. 26

4. **THE ITERATIVE DESIGN PROCESS IN THE SWITCH PROJECT** 27
   4.1 **PHASE 1: GATHER AND ANALYZE USER INFORMATION.** 27
   4.2 **PHASE 2: USER INTERFACE DESIGN** ..................... 32
   4.3 **PHASE 3: CONSTRUCT THE USER INTERFACE** ........ 48
   4.4 **PHASE 4: VALIDATE THE USER INTERFACE** .......... 49

5. **DESIGNING A GOOD USER INTERFACE** ....................... 51
   5.1 **BEING IN CONTROL** ............................................ 51
1. Introduction

This research project will look at various criteria for prototyping, designing and testing the best possible user interface for a presentation program. Several fairly new and uncommon components will be introduced into the GUI, and these components will hopefully aid the user in making and editing presentations faster and easier. It is a project goal to study how intuitive these components are for different types of users, and if possible redesign the user interface to better fit the users by using the findings from the research.

The presentation programs that exist today, and first and foremost Microsoft PowerPoint (figure 1), have more or less become a de facto standard [67]. A few others have tried to make their way into the market, but to little avail or only successful as targeting a smaller user group. The result is that users are stuck with few proper programs to choose from and a giant market leader that has seen little change over the last ten years.

![Figure 1: Screenshot of Microsoft PowerPoint](image-url)

Another, but secondary, project goal is creating more modern and stylish design templates. Due to limited graphics processing capabilities in the past, designs were usually restricted to
vector graphics, and consisted of simple gradients and solid fills. Today however, the situation is a different one. Processor power is far greater than five or ten years ago, yet the design templates have changed little. Good design is eye candy and relaxes the presentation impression. An appealing user interface will also to some degree encourage the user to use the program and to learn its abilities. The contrary, forcing the user to use something he or she doesn’t like because of how it looks or feels will probably make the user unhappy, uncomfortable, or at the very least discourage the user from using and learning the program. Aesthetics do count, no matter how well the user interface performs on the usability tests [3].

The project includes building a presentation program from scratch. To some extent, controls in Microsoft PowerPoint and other Microsoft Office programs, Macromedia Flash, SWiSHzone.com Swish, Microsoft Visual Studio, and Adobe Photoshop have been used as templates for parts of the user interface. This does not necessarily mean that the look and feel of these controls is the best and most intuitive ones, but to a certain degree the various users need to recognize and relate to the different elements in a GUI. The GUI principle called consistency is strongly related to this effect and will be described further in chapter 5. Examples of this are the button for making text bold and the button for printing (figure 2). The user friendliness of standardized buttons and tools are beyond the scope of this project.

![Figure 2: The buttons for making text bold and for printing are easily recognizable in most user interfaces.](image)

All program code files can be downloaded from:

1.1 What Is a User Interface?

The term “user interface” is one of those words we tend to hear a lot, but might not think much about. A definition from Wikipedia goes like this [71]:
“The user interface is the aggregate of means by which people (the users) interact with a particular machine, device, computer program or other complex tool (the system). The user interface provides means of:

- **Input**, allowing the users to manipulate the system
- **Output**, allowing the system to produce the effects of the users' manipulation”.

We use the term in a number of ways – both as a noun and as a verb.

Interfaces are all around us. When a person goes to the bathroom, the doors to the rest room are one interface to deal with. Another interface can be the look and feel of the toilet or the sink where people wash their hands.

Interfaces on similar items are rarely exactly the same. Doors look different and the flusher on the toilet might be in different locations. Luckily, our cognitive psychology relates similar items to each other and our brain will categorize the items we interact with. Imagine for a second how difficult life would have been if you would have had to relearn the entire usage of an item every time you encountered another model where the only difference was the color or the location of an insignificant detail. The fact that the human brain is indeed very good at generalizing and relating things with similar patterns can make the design process of user interfaces easier.

### 1.2 What Is a Computer Interface?

Working with a computer, a user will interact with both hardware and software. The hardware interface includes all hardware components that are connected to the system. This can be a keyboard, a pointing device like a mouse, a joystick or trackball, and includes the processing unit and the display screen itself. The software components of the user interface are all the items the users see, hear, point to, or touch on the screen to interact with the computer itself, as well as the information with which the users work [10]. Baecker et al defines the software user interface the following way:
Narrowly defined, this interface comprises the input and output devices and the software that services them; broadly defined, the interface includes everything that shapes users’ experiences with computers, including documentation, training, and human support [11].

1.3 Computers and User Interfaces

The computers of today utilize a new breed of software that revolutionizes the way people work. A popular belief is that these new systems make people’s lives easier and that their software experiences are better. Are they really? If all software products were as good as they were advertised to be, wouldn’t life be swell, work fun and problems nonexistent? Unfortunately, it turns out systems aren’t always as easy to use, intuitive and easy to learn as they are advertised. So why then is the look and feel of a computer program so important? How can we tell what the broad range of users really want or need? These are questions not easily answered, but one thing is still certain: the user interface is a key element in any good software solution.

Usually, a wide range of people are involved in creating software. Among these are for example software developers, interface designers, information developers and technical writers, help- and tutorial developers, usability professionals, project leaders and development managers [1]. A key element in creating good user interfaces is knowing the users. Needless to say, different users have different preferences and backgrounds. A successful piece of software typically needs to function in the intended environment, or risk failure in the targeted user group. The best user interface is the one that lets users do what they want to do, when they want to do it, and how they want to do it [1]. In spite of this, ironically enough, the third most difficult computing skill to find and hire for major companies in the United States was the GUI designer. At the fifth place on the same ranking list was the GUI programmer [2].

GUI design is often considered a trinity: function, aesthetics and performance [3]. When a user interface is evaluated, it’s usually its functionality that’s being judged. By far, functionality is indeed important, but still it’s only one part of the full story.

Aesthetics are how things are shown and presented. It is also the style in which things are communicated to the user. However, aesthetics is furthermore a very subjective thing to
measure and thus also hard to evaluate. Now, aesthetics can probably be viewed as almost as important as functionality. The reason for this is that the way a user feels about a product will to a large degree influence the way he or she works with it, the ability and willingness to learn etc. Aesthetics is not necessarily about choosing the colors of the application, the number of artwork images, or the flashing graphical animations. It’s rather about how well the application conveys the message intended [3]. Programmers can sometimes trick themselves into believing that the more show-off the interface is, the more impressing it will be for the users and that it will convince the world how talented they are rather than actually aid the user in doing something [38].

The last part of the trinity is performance. Performance means speed, trustworthiness and durability. A well designed application should be swift to work with, convey reliability by not crashing or contain other errors, and probably not be too processor intensive.

1.4 A Brief History of User Interfaces

Like many developments in the history of computing, some of the ideas for a computer with a graphical user interface were thought of long before the technology to build such a machine was available. The American scientist Vannevar Bush envisioned in the early 1930s a device called the Memex, which would look like a desk with two touch screens, a keyboard and a scanner [69]. The machine would allow the users to access human knowledge through connections very similar to hyperlinks on an internet. Nonetheless, at this point the digital computer had not yet been invented so Bush’s ideas were not widely read or discussed at the time.

Starting in about 1937, several groups of scientists around the world started constructing digital computers. World War II provided much of the necessary motivation and funding for producing programmable computing machines for everything from calculating artillery firing tables to deciphering enemy cryptographic codes. With the perfection of vacuum tubes and commercial production was started, the fast switching mechanisms the computers needed was a reality. Nonetheless, another twenty-something years would pass before the
thoughts of a graphical user interface would be revisited. Douglas Engelbart\(^1\), sometimes referred to as “the father of the GUI”, was an electrical engineer with a background as a radar operator during WWII [69]. In his 1962 essay “Augmenting Human Intellect” [70] he argued that digital computers could provide the quickest method to “increase the capability of a man to approach a complex problem situation, to gain comprehension to suit his particular needs, and to derive solutions to problems.” He envisioned the computer as a tool for enhancing human intellect, and described a hypothetical example similar to modern graphical CAD\(^2\) systems. This was a huge leap in thinking for his time. Computers in 1962 were giant mainframes where programming (batch processing) consisted of punching out holes in a series of punch cards that were then handed to a computer operator for processing. The results could typically be picked up hours or days later. Douglas and his growing staff worked for years on a technology that finally culminated in a public demonstration for more than a thousand computer professionals in 1968 [12].

Generally speaking, the history of user interfaces can be divided into the following eras [71]:

- Batch interfaces (1945 - 1968)
- Command-line user interfaces (1969-1983)
- Graphical user interfaces (1984 - present)

Batch processing is the sequential execution of a series of programs (“jobs”) on a computer. The early batch interfaces came from back in the days when programmers wrote code on paper forms that were then translated to keypunched 80-column punch cards or paper tape. The cards or tape would then be passed to a system operator who would schedule the execution. Once it was time to run the program, the system operator would feed the program tape or cards into the computer. Since a number of operations would be “batched” together instead of being run immediately, the tasks were referred to as “batch jobs” and the interfaces thereafter.

Command-line user interfaces (CUI) or a command-line interface (CLI) is a way to interact with the computer via a text terminal. Commands are entered as lines of text and output is

\(^1\) Douglas Engelbart was also the scientist whose team developed the mouse and the hypertext.

\(^2\) Computer Aided Design, 3D graphics programs used for engineering, visualization, simulations and more, usually within architecture, physics and construction.
accordingly lines of text. The command line interfaces originated back in the 1950s when teletype machines were connected to computers and represented a large step forward from the previous punch cards or paper tape. The command-line interfaces are closely related to the concept of timesharing computers where perhaps the most influential experiment was MULTICS in 1965. In 1969 UNIX was born, and the creation of this operating system marks a milestone in computer history. Today, the command line still exists in most operating systems and some computer users continue to live in the superstition that the command line lurks behind the scenes secretly controlling things [13]. One very typical feature of the command-line interfaces is that there is no *undo*.

There were several sporadic experiments with graphical user interfaces as far back as in 1962. Those days, displays (monitors) were monochrome character terminals that usually displayed nothing but characters and command-line text. The imaginative researchers came up with an original solution. Instead of a character terminal the display was a modified oscilloscope and was connected to a PDP terminal. The first program made for this interface was a game, it was released in 1962 and it was called SPACEWARS [99] (figure 3).

![Figure 3: The first graphical user interface, the computer game SPACEWARS connected to a PDP-1 computer through an oscilloscope. The two big white lumps are the spaceships trying to kill each other.](image)

Another huge impact on the development of GUIs was caused by Ivan Sutherland’s *Sketchpad* [73]. As his master’s thesis in 1963 he used MIT’s $10 million machine and created a system for direct manipulation of objects using a light pen [74]. His ideas spawned the very first seeds for iconic representations, object oriented techniques, constraints, interaction techniques and approaches to animations [73].
It was readily understood already at this time that graphical user interfaces would make a compelling user experience. A few years later, in 1968, Douglas Engelbart and his team from SRI demonstrated the first computer mouse and hypertext, even though it turned out to be video games that mass marketed the graphical user interfaces since they ran hardwired programs on extremely cheap and simple processors. Ordinary computers cost hundreds of thousands of dollars at this time, and oscilloscope displays became an evolutionary dead end. Inspired by Engelbart’s 1968 demo, in 1973 researchers at Xerox Paolo Alto Research Center (PARC) developed a machine called the Alto [75]. It featured a bit-mapped monochrome display\(^3\) and a mouse (figure 4).

![Figure 4: The first GUI on an Alto workstation, displaying the file browser. The image has been clipped at the bottom.](image)

The crude and modern GUI of the Alto turned out to become the predecessor of the coming GUIs in the time ahead. It featured all the logical components of GUIs as we know them — icons, windows, scrollbars, sliders, and the like. Nevertheless, the Alto remained an experimental project [76]. In 1981, the first commercial machine with a GUI hit the market. The Xerox 8010 ("Star") was based on the Alto, but never became very widespread. GUI computers were failing because they were either priced too high or too weak to devote enough processor power to justify a GUI [73]. In fact, the HCI community at the time believed the GUIs presented a disadvantage since they distanced the users from the underlying systems, which many believed experienced users had to learn eventually [73].

---

\(^3\) A bit-mapped display addresses each pixel on the screen individually, rather than assigning a character to a location.
Apple Lisa by Macintosh, released in 1984, was the first commercially successful product to use a GUI. For this reason the era of GUIs starts this year. The first version of Windows, Windows 1.0, was released in 1985 [72].

1.5 A Brief History of PowerPoint

The history of PowerPoint runs as far back as 1984 [78]. Former Berkley student Bob Gaskins envisioned an easy to use program that could manipulate a string of slides. He teamed up with a failing Silicon Valley company called Forethought and created Presenter. This name was soon changed, however, due to a trademark issue. The new name was PowerPoint.

In 1987 PowerPoint 1.0 for Apple Macintosh was released. It was purely black-and-white and produced text and graphics slides that were easily transferred to a printer or photocopier as transparencies or hand-outs. Later that year a full color version was developed to accompany the first color Macintoshes that appeared in the market and had a special 256 color graphics card installed [77].

Parallel to this, a company called Genigraphics had begun working with Microsoft on a similar program. Seeing little threat in Forethought, their program enabled authors to send their files to Genigraphics for production of high quality 35 mm slides. The developers at Genigraphics had spent years creating layouts for big corporations and many of these templates are the same we still find in PowerPoint today [77].

In 1987 Forethought and PowerPoint were purchased by Microsoft for $14 million [78]. As PowerPoint advanced, Genigraphics suffered, and the leaps in technology that enabled the users to produce their own presentations on transparencies slowly led to the demise of Genigraphics. In 1994 Genigraphics filed for bankruptcy, but was rescued from going under by InFocus Systems, a large LCD projector manufacturer [77].

In 1988 the first Windows and DOS versions of PowerPoint were produced [78]. PowerPoint was envisioned to become a huge hit and has since 1990 been a standard part of the Microsoft Office suit of applications. Being a part of Microsoft Office has made PowerPoint the most widely used presentation program on the planet [78]. A key feature such as being able to share PowerPoint files across different platforms like Macintosh or between users has
been an important factor in its success. Today, even competing presentation programs such as Apple KeyNote or OpenOffice.org Impress add support for reading and converting PowerPoint files. However, due to the strong relationship between PowerPoint and the Windows platform, and then again largely because of the possibility to embed content media as OLE objects\(^4\), some presentations cannot be shared across platforms. The PowerPoint file format is kept secret and reading and writing of the file format happens through platform specific operating system objects. This has lead to a movement towards more open standards such as PDF and OASIS [78].

\(^4\) OLE is an acronym for Object Linking and Embedding.
2. Research Problem

This study will focus on the prototyping of a graphical user interface for a presentation program. Now, it seems user interface testing and development is a well researched area. The big software companies are well aware of the importance of a good user interface and conduct many usability tests before applications are released. However, there’s an inevitable distinction between creativity and following the mainstream. In fact, many interface projects are sidetracked because of the developers’ needs to be creative [4].

Windows developers are said to have it rather easy. A key ingredient to making a program that the average user can understand, and even enjoy, is usually just to copy Microsoft’s products as closely as possible [4]. This doesn’t necessarily imply that Microsoft has made all the right decisions in their user interfaces. However, that’s not really important. If the users that are the target group of the application have ever used a computer before, chances are that it’s been Microsoft Word or Microsoft Internet Explorer on Microsoft Windows. In fact, if the users are regular computer users, they probably spend most of their computing time with Word or Excel.

There’s rarely a good reason for deviating far from Microsoft standards [4]. An average user is, if anything, probably familiar with the common keystrokes and menu organization of the various Office programs. Yet, even though Microsoft is known for putting huge amounts of money into their usability tests, an old saying goes “only dead fish follow the stream”. Other software companies have successfully built user interfaces that don’t conform to Microsoft standards, and it’s probably these companies’ inventiveness and innovation that keeps Microsoft from being a standard rather than guideline. Examples of these are the palette panels of Adobe Photoshop (figure 5) and Macromedia Flash (figure 7) or the highly inventive and prize winning interface of Kai’s Tools (figure 6).
Figure 5: Screenshot of Adobe Photoshop

Figure 6: Some of Metacreation Kai's Power Tools' interfaces
2.1 The Core Research Problem

This thesis will attempt to dive into the possibilities and limitations of graphical user interface design. Without breaking the limits and rules of common conventions, explorations of fairly new elements will be introduced and tested. An example of this is the highly customizable PropertyGrid (figure 8), introduced by Microsoft with the Microsoft .Net SDK. The PropertyGrid allows for ordered attribute-value pairs, structured into categories, and with optionally custom UITypeEditors. A UITypeEditor is a floating panel (usually small) with user interface elements like buttons or a drop-down list. Such an editor makes the selection of the correct value easier, usually more logical than text values, and can restrict the user from entering wrong values. See figure 9 and figure 10 for examples.
Figure 8: A typical PropertyGrid. It’s easy to make out the attribute-value pairs, and ‘Behavior’ is a category containing more such pairs.

Figure 9: An example of a custom UITypeEditor. The slider (marked in red) is not a native windows component in the PropertyGrid and will pop up when the user clicks inside the transparency value field. Such components can enable values to be specified in a more intuitive manner.
At the core of this project is the prototyping and development of a presentation program with an intuitive graphical user interface. Prototyping itself involves all stages from starting a design with sketches on paper until a fairly usable product is produced. This in turn involves interface research, different stages of usability testing, and the final product may require many iterations. The prototyping phase will cover some of the important phases in product development, but due to the size of the project it will not cover all the phases up to the final product. As for Switch, the iterative development process is described in chapters 3 and 4, and the software usability testing is described in chapter 7.

Besides preliminary research, the development phase is quite complex and consists mainly of two parts. The first part is the development of the user interface itself. Not only is this a complicated programming task, but it also depends on making a wide range of design choices. In many respects it also involves making a wide range of choices for the user. An evaluation of the different aspects of the theory and practice of this development process can be found in chapters 5 and 6.

The development of custom components will be a very central part of the development of the user interface. Only a limited amount of possibilities lie within standard components such as
list boxes, combo boxes and buttons. Quite often, custom controls need to be built. A custom control can be a button, checkbox, radio button etc with increased or altered looks and functionality. Such functionality can be informative mouse-over effects, custom drawing of a control\(^1\), special handling of elements in a list box, and so on. The slides list is an example of such a custom control. Standard list boxes cannot contain images. For this reason, the painting of the elements of the list must be overridden to allow for the drawing of the images of the slides. Some of the custom controls developed are presented in chapter 6.

As with any presentation program, the program needs to produce reliable presentations. This introduces the second major problem of the development process, namely the development of a library capable of producing swift and reliable files. Flash (SWF) is a proprietary file format from Macromedia and has been chosen as the output format for a number of reasons. First of all, the players are readily available. Nearly all browsers in use today have a flash player plug-in installed\(^6\), and the files can also be played in standalone players available on many platforms. Furthermore, the format is efficient. By taking advantage of underlying hardware when it’s available, the players can render the graphics very fast. A third point is interactivity. Flash has been developed with interactivity as a goal\(^{54}\) and the players already support many ways for the user to interact with a flash file. In addition to this, the format is extremely compact. It takes advantage of nearly every bit available and is thus ideal for web graphics. There is also a matter of recognition in the output file presentations since the files have a distinct look and feel, and finally the library will serve as a base for a versatile library in future work.

The SWF file format is very complex and highly compact. As far as searching on the internet has shown, such a library has up until now not yet been developed anywhere using the C# programming language. Since the C# programming language is quite high-level and produces executable files compared to Java bytecode, it cannot make use of many of the advantages that lower level programming languages have. One major challenge besides actually producing the correct output file format is thus speed. The development of the SWF library is covered in chapter 8.

---

\(^1\) Custom drawing of a control, or more technically owner-drawn controls, is the procedure of altering the appearance of a control at runtime. See chapter 5 for more information.
GUI-design is a trade-off. It’s a trade-off between possibilities and user-friendliness. By restricting the user’s choices, one is at the same time removing elements where the programming could go wrong. However, some developers do the opposite. They believe that if it’s difficult to decide between two ways to provide a feature, why not provide both and make the user decide? Unfortunately, this type of logic (deciding not to decide) is probably not ideal at all. The end user might be confused by even more options, it raises the number of possible problems and almost guarantees a lack of consistency across different installations.

The GUI designer must know the user. Sometimes a program needs to provide lots of options and sometimes the best way to handle things is to narrow down the options to an absolute minimum. Imagine for example a quite complicated device such as a microwave oven. The user is normally presented with options like duration, power, start and stop. Now imagine for a second in an extreme example what would happen if the user had the possibility to change the pitch of the “food ready” beep, the length of the beep, the intensity of the interior light, the time display mode and so on. Every time the user wanted to use the microwave, he or she had to navigate through a series of menus where only a fraction of the options were of real interest. Obviously, these added features would reduce the user friendliness of such an interface and would probably not interest the average user. Sure, it could be nice having the opportunity to adjust the platter rotation speed, but most of the times these options are better off handled by a reasonable default, removed altogether or hidden away in a prominent place where the advanced user can configure them. Remember, each time the user is presented an option, he or she is at the same time being forced to make a decision. Many users become increasingly unsettled and less confident as they have to make decisions they don’t understand.

This project, however, tries a slightly unorthodox approach. First of all, all program element properties (like a textbox in a presentation) are available in the property grid for quick editing. This puts all object properties in one ordered list available at any given time. Usability testing will try to evaluate the user friendliness of the property grid.
Even though the property grid is new to many users, Microsoft has been using it for years in Visual Studio .Net. Nevertheless, Visual Studio is a tool for programmers and programmers tend to have a better way with programs than most common people. Deciding on what information should be included in the property grid – or if it should be there at all – is a goal in this project.

The approach is unorthodox because the project includes the usual edit properties dialog-windows as well. To not deviate too far away from Microsoft de facto standards, the user still has the opportunity to right click on an object and get the properties in a modal window. Tests in chapter 7 try to explore which approach is better.

### 2.2 Limitations

This project can only test a small part of user interface usability. Even though only a few elements are tested and the tests focus on aspects of a presentation program, this project still covers a substantial research area. Hence, several parts are not covered here. For once, the usability tests are restricted to the elements of a presentation program. This is because the program to be tested is the core of the project. Secondly, the testing focuses on mainly three elements of the user interface: the slides list, the property grid and the main content panel. Herein lays the main functionality and almost all attributes that can be changed by the user.

The project does not test customization and tailoring. Customization and tailoring are considered research fields of their own and have already been covered in other theses. Yet, an interesting follow-up to this thesis could look at using the property grid for such purposes.

The project does not cover installation. The usability testing phase assumes the users arrive at a computer with a preinstalled, fully functioning version of Switch.

This thesis does not dig deeply into the wide research area of help systems. Usability testing of different approaches of implementing “help” is simply too big to be included in this

---

2 A window that appears on top of the running application and the user cannot return to the application until a choice has been made or “Cancel” is pressed.
project. The project does, however, look briefly at “tool tips” and the “info” function implemented in the PropertyGrid described in chapters 1 and 5.

Localization and internationalization are two other areas that are not be covered in this project. Their definitions are adjusting a program to a user’s locale by making it possible to change language, so that the application automatically adjusts to the time zone, that the right currency symbols are used and so on. Some countries use comma as a decimal separator, some countries use a punctuation mark. Some places use AM/PM notation for time, others use a 24 hour format, etc. These are typical issues in this context and are implemented as far as possible, but have not been prioritized or covered in the analysis.

The presentation program output format also contains a user interface. This interface is a result of the program elements the user selects to put in the presentation. The thesis focuses on the main program user interface and does not cover the output file format user interface.

A presentation program should contain a drawing module. Thus, a simple drawing module has been implemented in the user interface. The user is able to create rectangles, ellipses and lines with varying colors and fills. However, this is not a drawing application. There are far better specialized tools out there for drawing and imaging and this program only contains a little subset of all those advanced features of other programs. The target user group is basically anyone with basic computer knowledge, and advanced drawing programs are not common commodities in this target user group.

Possibilities to import files of other formats are a key feature in modern software. This project does not add support for import and export of multiple formats. Import and export of PowerPoint files is planned for future releases.

It is no secret that there are only a very few other commercially commonly available presentation programs out there. However, this project does at no times offer to make a comparison between Switch and these other programs.

The development process could have benefited from access to substantially different presentation programs, such as Apple KeyNote. Such access could thus have provided insight into how different products have solved similar problems. Unfortunately, these other programs have not been studied and thus PowerPoint has been the major source of information for such development.
2.3 Motivations

This project is motivated by several factors. First of all, hours of frustrating struggles against PowerPoint, always having decisions made for oneself, only to correct them followed by a new autocorrect shortly after has been extraordinarily annoying. In this context the famous saying “less is more” gets a clear meaning. Sometimes, adding tons of code for making a program “super smart” might not be the best approach. By not treating an average user as completely inexperienced, Switch will try to show that ordinary users can make the proper decisions if the interface is intuitive enough.

Another motivation is to create something that is highly portable. Microsoft PowerPoint, for instance, is very tied to the Windows platform. When a user tries to send a presentation to another platform such as Linux or OSX, the presentation might behave weird or not work at all. Especially if a presentation has embedded Windows-native OLE objects, things are often bound to crash or not work on other platforms. Even fonts sometimes get a whole new look when a presentation is moved from one machine to another since they are commonly based on font names. If for example two different machines have the same font installed, but under different names, or two different fonts are installed under the same name, weird things are not uncommon. This is not an unusual situation when presentations are ported between machines and platforms. Different hacks have been developed on other platforms to partially compensate for this, and Mac has its own version of Microsoft PowerPoint. On Linux you can use the similar OpenOffice presentation program. Yet, most of these problems can be solved by turning to a third party readily available open source file format.

Switch will produce presentations in the Macromedia Flash proprietary format. Available on more than 90% of browsers used on the Internet today [6], it’s probably just as widespread as any PowerPoint reader. The format can embed fonts, images, sounds and movies, thus omitting the portability issues altogether. Presentations can be viewed in a browser with the appropriate plug-in or in a free standalone player. The animations are fast, elegant and can be embedded in a web-page, certainly a good starting point for an interesting project. Target users are anything from the average users and newbies to advanced users, web designers and developers.
3. Research Design and Research Methods

Scientific research is commonly subdivided into qualitative and quantitative methods. The two forms both contrast and complement each other and the nature of the research area will define whether one or the other, or both, forms should be used. The chosen form of research will then in turn be conclusive for the kind of data collected.

Qualitative methods involve an in-depth deeply elaborated approach. Patton defines qualitative research as a naturalistic approach that seeks to understand phenomena in context-specific settings, such as "real world setting (where) the researcher does not attempt to manipulate the phenomenon of interest" [82]. Broadly defined this involves any kind of research that produces results that are not a product of statistical procedures or other means of quantification [83].

Quantitative research, on the other hand, involves heuristics and collection and statistical analysis of empirical data. The method is mainly about classifying phenomenal features, counting them, and even constructing complex statistical models as an attempt to explain the observations [84].

Table 1 is taken from James Neill and explains some of the features of qualitative and quantitative analysis methods [86].

<table>
<thead>
<tr>
<th>Qualitative</th>
<th>Quantitative</th>
</tr>
</thead>
<tbody>
<tr>
<td>The aim of <strong>qualitative</strong> analysis is a complete, detailed description.</td>
<td>In <strong>quantitative</strong> research we classify features, count them, and construct statistical models in an attempt to explain what is observed.</td>
</tr>
<tr>
<td>Recommended during earlier phases of research projects.</td>
<td>Recommended during latter phases of research projects.</td>
</tr>
<tr>
<td>Researcher may only know roughly in advance what he/she is looking for.</td>
<td>Researcher knows clearly in advance what he/she is looking for.</td>
</tr>
<tr>
<td>The design emerges as the study unfolds.</td>
<td>All aspects of the study are carefully designed before data is collected.</td>
</tr>
<tr>
<td>Researcher is the data gathering instrument.</td>
<td>Researcher uses tools, such as questionnaires or equipment to collect numerical data.</td>
</tr>
</tbody>
</table>
Data is in the form of words, pictures or objects.  
Qualitative data is more 'rich', time consuming, and less able to be generalized.  
Researcher tends to become subjectively immersed in the subject matter.

Data is in the form of numbers and statistics.  
Quantitative data is more efficient, able to test hypotheses, but may miss contextual detail.  
Researcher tends to remain objectively separated from the subject matter.

<table>
<thead>
<tr>
<th>Table 1: Features of qualitative and quantitative research methods.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Qualitative Methods</strong></td>
</tr>
<tr>
<td>Data is in the form of words, pictures or objects.</td>
</tr>
<tr>
<td>Qualitative data is more 'rich', time consuming, and less</td>
</tr>
<tr>
<td>able to be generalized.</td>
</tr>
<tr>
<td>Researcher tends to become subjectively immersed in the</td>
</tr>
<tr>
<td>subject matter.</td>
</tr>
</tbody>
</table>

There is an ongoing debate of whether quantitative or qualitative research methods yield the most reliable and valid results. However, over the past decade there has been increasing trend of blending quantitative and qualitative methodologies to provide a broader and deeper perspective [87]. Although most researchers prefer to conduct either qualitative or quantitative research, some researchers have suggested combining the two methods. This is then called triangulation [84].

At this point the research methods used in the Switch project are based on qualitative research. This coincides with what Neill says above [86] since the project is still in “earlier phases”. Creswell furthermore notes, “An individual begins with a theory, collects data that either supports or refutes the theory, and then makes the necessary revisions before additional tests are conducted” [7] [5]. This approach is fairly descriptive for any iterative development process and will also be followed in this project.

### 3.1 Qualitative Methods

GUI design and testing is difficult and laborious. Most users have different preferences and experience, and testing takes time. The different elements of the user interface need to be tested individually to find out exactly what parts work, how well they work, and what parts don’t.

Jacob Nielsen has proposed Heuristic Evaluation as one of the preferred methods for finding problems during user interface research [88]. It involves having a small set of evaluators
examine the interface and judge how well it conforms to recognized usability principles (the “heuristics”).

The qualitative approach in this project has a very iterative nature. It is based on iterating research, design, and evaluation and is explained in detail in section 3.3 and chapter 4.

Using a usability lab at the Institute of Informatics, University of Oslo, users can interact with a computer and at the same time be observed by a camera. The qualitative evaluating approach is hence based on a short introductory interview, followed by a recorded session in front of a computer where the users interact with the interface, followed by another interview. During the testing process, the users are asked to think out loud and explain the reasons for every step they make. How the different users interact with the program is a key to understanding the usability of the different elements. The users will also have a series of assignments to complete and the tests uncover how well the user interface conveys its functions. Unfortunately, such research takes long time. Time only allows for a limited amount of users to study and is hence a critical factor in this research. The number of users studied will to a large degree affect the results, yet chapter 7 argues for that adding more users to test a flawed design will only yield the same results.

Another part of the qualitative approach is conducted by interviewing a usability expert and cognitive psychologist. Human-computer interaction (HCI) specialists have proven to uncover more problems in designs than many other professionals. Chapter 7 will describe this research in more detail.

3.2 Quantitative Methods

An approach that could have added a lot to the research is a quantitative approach. If the users had been allowed to test the program e.g. by downloading it online, they could have answered an online questionnaire afterwards. This way a large amount of users could easily have been tested at their own pace, but at the expense of in depth interviews and thorough understanding of every testee. Nevertheless, the questions for such an approach would have needed to be the subject of research on its own as the questions would need to cover all personality types and previous experience. The questions would perhaps even be needed automatically adapted to each test case depending on certain rules, such as e.g. previous
experience. The development of a quantitative approach in addition to the qualitative one would probably have been the best approach. Unfortunately, the quantitative approach will not be implemented at this point due to implementation time.

A somewhat similar approach that deserves mentioning in this context is the approach used by IBM when they developed numerous information kiosk systems for the 1984 Olympics and the 1991 World Games in Barcelona [34]. The IBM development team installed an information kiosk in the lobby of the IBM building, and attached a notebook where people who tried the information kiosk could write notes about what was bad about the design and what was good. For the same reason as above, time does not allow for such an implementation.

An important point in quantitative research is nevertheless to get a wide range of people to try to use the product, not just a small preselected group of test participants.

### 3.3 A Four Phase Interface Design Process

The development of the user interface may or may not be a part of the product development process. However, what separates the development of the user interface from other areas of software development is the focus of the development. The user interface development focuses on the interface elements and the objects the users perceive and use, whereas other parts of the development process might focus on e.g. functionality. Both the development of the functionality and the development of the user interface are to some degree separate processes and can proceed in parallel, especially in the early stages. Later in the development cycle, feedback from usability tests and user interface concerns should drive the program design [20].

The user interface development is a cyclic process. Validation ensures what needs to be changed and redevelopment brings the end result closer to the ideal interface. The design process is hence also iterative by nature. Iteration is continued for as long as necessary\(^1\) until the desired results are produced. The four major phases in the process are [20]:

\(^1\) “Necessary” is a highly subjective term, and can mean anything from “good enough” to “until time (or money) runs out”.
Phase 1: **Gather and analyze** user information.

Phase 2: **Design** the user interface.

Phase 3: **Construct** the user interface.

Phase 4: **Validate** the user interface.

This process is illustrated in figure 11.

![Diagram](image)

*Figure 11: An iterative user interface design and development process [20].*

Figure 11 could have been depicted in numerous ways. Some authors draw the figure as an inward or outward spiraling process, where the point is to get closer and closer to the goal throughout the iteration process. The point is, however, that refinement is an ongoing process [20].

Also the Windows 95 user interface style guide describes a similar model for user interface design [16]. According to Microsoft, the design model should be based on three phases: Design, prototype and test. They also state the importance of gathering a design team with diverse backgrounds. There should be a good balance between the different areas of research the different members of the team possess. [16]

Traditional development methods often follow a “waterfall” process. This process is well covered elsewhere [17], and includes the similar phases as described above. The main difference, though, is that this process is considered *linear* rather than iterative. All software development methodology today should take advantage of the concept of iteration where this is possible.
It’s a common question to ask “how long should the iteration process continue?” The general answer is “it depends” [20]. The project may run out of time, resources or money to further iterate the design. Ideally, the product criterions set beforehand are met throughout the process and the validation phase shows that users’ performance data and subjective ratings of the product meet or exceed the product’s planned goals and objectives. It’s not uncommon that in the order of 70% of product cost is spent on user interface design [18].

3.4 Research Team Members

The creation of a user interface is a complex process. It involves functionality, programming, design, graphics, human factors, psychology and much more. In a real world scenario, a design team where the individual members possess these skills would work closely together to create and test the user interface. Baecker et al. says it the following way [14]: “Interface design and development require software engineering and programming skills, of course, but can also benefit from the skills of graphic and industrial designers; human factors engineers, and psychologists who understand human cognitive, perceptual, and motor skills; technical writers and training specialists’ people knowledgeable in group and organizational dynamics; and those with expertise in input devices, display technologies, interaction techniques, dialogue design, and design methodologies.... The growing use of sound, voice, video, animation, and three-dimensional display draw upon still other specialties.”

Interface design hence involves many skills. Since the skills required are so diverse, and the task so complex, many interface design experts claim that no one team member will possess all the necessary skills to design both the user interface and the product code [15]. Some might be knowledgeable in more than one field, though. Also the Windows Interface Design Guides make a point out of gathering a design team with different backgrounds [40].

The very fact that so many skills are needed has been a challenge in this project. To take on so many different roles is indeed no easy task, but gives at the same time a large amount of insight.
4. The Iterative Design Process in the Switch Project

When designing a large scale program, one must first gather information about the users and their environments. There is a good chance the users are currently using a similar program today that performs the task or part of it. Maybe several programs together perform the desired task. There are many ways to obtain this information, and user interviews, surveys, observation and perhaps videotaping might be the proper way to do it. The questions the users should be asked must be well thought through and the analyzing afterwards equally careful. Borenstein (1991) says: “Listen to your users, but ignore what they say.” [19] Users have a tendency to like whatever they’re currently using even though they would still admit it could be better. Since users normally are not up to date on the latest technology, it can be difficult to have them break free of their current mindset [20].

4.1 Phase 1: Gather and Analyze User Information.

The natural place to start is the users. Phase one of the iterative design process is to gather and analyze user information. This is done by defining the problems the users want to be solved and figure out how they do their jobs. It’s important to learn about the users’ capabilities and the tasks they perform.

The user information gathering task can be broken down into five steps [20]:

1. Determine user profiles
2. Perform user task analyses
3. Gather user requirements
4. Analyze user environments
5. Match requirements to user tasks

To determine the user profiles, one needs to ask: Who are the users? This includes demographics, skills, knowledge, background, and any other relevant information that is essential to describe the user information. User interviews, surveys, observation, video
recording, industry reports, reviews, and press- and marketing materials can be good information sources.

For the case of this project, almost any person might be a probable user. An obvious prerequisite is basic computer experience on the Windows platform. With the knowledge that today’s computer users span all ages and both sexes, gender and age will not affect the choice of candidates. Since this version of Switch is written in English, English-language skills are necessary.

It might be useful to divide the user groups into two categories. The first is the so called *novice users*. These have the basic skills to use a computer, but might not know all the shortcuts, the average program capabilities\(^1\), but still have the most basic knowledge of common Windows programs such as Word or Excel. A second user group can be defined as *advanced users*. These users have used, and are fairly familiar with Power Point, have created slideshows and will quickly recognize a typical Microsoft Office-style user interface. The results from this and the following four steps are summarized in table 2.

Secondly in phase 1, one needs to analyze the user tasks. This involves what users want and how they accomplish their tasks. It’s also important to identify the most critical tasks, the common workflow, the steps the user need to complete the desired tasks, how frequent the different program elements are used, and so on. Naturally, this step can be very elaborate. It all boils down to how detailed the analysis need to be. For the case of the Switch project, the users need to be able to create a slideshow and edit its properties, create, edit, delete and rearrange slides, navigate the slides menu, and edit the slideshow text- and image elements, and their properties.

The different user groups in this step will probably perform very differently with respect to time used. With the most common Windows Office shortcuts available, advanced users might use this to their advantage to work faster. They might also be less afraid of doing things that can “ruin something on the computer” – a common misconception many novice users have. They will certainly familiarize themselves faster with the user interface.

---

\(^1\) Average program capabilities are such as the ability to copy, paste, undo, drag and drop, etc.
Advanced users will also probably learn faster because of their ability to explore the parameters of the different elements.

The third step in phase 1 is to gather user requirements. This typically means answering the question “what do the users expect the product and interface to do for them”. In the case of software development as a hired assignment for an organization, such information is commonly gathered through the use of focus groups, structured interviews and user surveys. Since this project aims at developing a user interface with a faster, cleaner and more intuitive user interface than the only well-known alternative, PowerPoint, a natural question to answer would be: “How is Switch going to improve the way the users work compared to their current situation?” Since the user interface is aimed at being swifter to work with, speed is an obvious requirement. For novice users, another goal will be a low-threshold learning curve because of the intuitive elements in the user interface.

Arriving at the fourth step in the first phase, it’s time to analyze the user environments. This involves looking at the environmental characteristics that may impact how the users work and generally answers the question “where do the users perform their tasks”. The elements to analyze are such as the physical work environment, user location and mobility[^2], human factors ergonomics and physical considerations[^3], users with special needs[^4] and internationalization and other cultural considerations[^5]. Naturally there’s a big difference between creating a system for an office worker than for a surgeon in a hospital operating room or a critical airline control system. Luckily, the operating system comes with a wide range of built in features for adopting the system to different environments. Switch is a typical office application and location can be solved using standard methods. Accessibility is covered with built-in operating system features, and the current version of the program will be in English only.

The final step according to Mandel [20] is a reality check. If the system is an order by another company, the user requirements and expectations might not be realistic. This is

[^2]: Office, home, on-site, mobile, FTP-server, etc.

[^3]: Vision, hearing, sitting/standing, keyboard abilities, usage of pointing devices, etc.

[^4]: Accessibility, and physical, cognitive, speech, and other disabilities.

[^5]: Internationalization is the process of translating a program to another language, character set, direction of writing etc.
where the system upcoming planned abilities and user requirements are set side-by-side and compared. If the users request text-only abilities, there is no need to implement a full multimedia platform.

In this project, the users are typical average users of standard Windows applications. However, this is not a custom ordered application so it’s difficult to compare user tasks and requirements. Instead, more thought will put into step 2, the analysis of user tasks. The iterative nature of the development process will ensure the best possible result since the tasks and the user interface is refined over time. It’s furthermore not uncommon that user requirements change during the design and development process. Usability is more thoroughly covered in the usability testing chapter (chapter 7).
<table>
<thead>
<tr>
<th>Analysis Results</th>
<th>Novice Users</th>
<th>Advanced Users</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Step 1:</strong></td>
<td><strong>User Profiles</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Female and male</td>
<td>• Female and male</td>
</tr>
<tr>
<td></td>
<td>• Any age</td>
<td>• Any age</td>
</tr>
<tr>
<td></td>
<td>• English speaking skills</td>
<td>• English speaking skills</td>
</tr>
<tr>
<td></td>
<td>• Basic Windows experience</td>
<td>• Basic Windows experience</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Experience with Microsoft Office, in particular Microsoft Power Point</td>
</tr>
<tr>
<td><strong>Step 2:</strong></td>
<td><strong>User Tasks</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Create a new slideshow</td>
<td>• Create a new slideshow</td>
</tr>
<tr>
<td></td>
<td>• Edit slideshow properties</td>
<td>• Edit slideshow properties</td>
</tr>
<tr>
<td></td>
<td>• Create slides</td>
<td>• Create slides</td>
</tr>
<tr>
<td></td>
<td>• Edit slide properties</td>
<td>• Edit slide properties</td>
</tr>
<tr>
<td></td>
<td>• Rearrange slides</td>
<td>• Rearrange slides</td>
</tr>
<tr>
<td></td>
<td>• Delete slides</td>
<td>• Delete slides</td>
</tr>
<tr>
<td></td>
<td>• Create a textbox</td>
<td>• Create a textbox</td>
</tr>
<tr>
<td></td>
<td>• Edit/delete a textbox</td>
<td>• Edit/delete a textbox</td>
</tr>
<tr>
<td></td>
<td>• Edit textbox content</td>
<td>• Edit textbox content</td>
</tr>
<tr>
<td></td>
<td>• Create an image box</td>
<td>• Create an image box</td>
</tr>
<tr>
<td></td>
<td>• Edit/delete an image box</td>
<td>• Edit/delete an image box</td>
</tr>
<tr>
<td></td>
<td>• Change image box content</td>
<td>• Change image box content</td>
</tr>
<tr>
<td></td>
<td>• Save the slideshow</td>
<td>• Save the slideshow</td>
</tr>
<tr>
<td></td>
<td>• Play the slideshow using the program controls</td>
<td>• Play the slideshow using the program controls</td>
</tr>
<tr>
<td></td>
<td>• Locate and play the slideshow using the Macromedia Flash standalone Player</td>
<td>• Locate and play the slideshow using the Macromedia Flash standalone Player</td>
</tr>
<tr>
<td></td>
<td>• Play the slideshow in fullscreen mode</td>
<td>• Play the slideshow in fullscreen mode</td>
</tr>
<tr>
<td></td>
<td>• Print a slideshow</td>
<td>• Print a slideshow</td>
</tr>
</tbody>
</table>
### Step 3: User Requirements

- Easily recognizable user interface elements
- Requires little or no training
- Fast to use
- Able to create, open, view, and print graphic slides
- No other tools/programs needed
- Undo-function

### Step 4: User Environments

- PC – Standalone program
- Usable at home, office or any common work environment using a laptop
- Windows XP with .Net environment installed
- Macromedia Flash Standalone Player installed

### Step 5: Matching Requirements to User Tasks

- Usability testing
- Find out whether what the users thought they wanted was what they really wanted

**Table 2: Results from the gathering and analyzing of user information phase.**

4.2 Phase 2: User Interface Design

The second phase of the four phase iterative design process includes a number of steps that are meant to be completed in sequence. Many programmers fall for the temptation of starting to code the full product without analysis up front. Theo Mandel opposes this idea with the following words: *How can you tell if a program works if you haven’t even decided what is a usable system? It’s like drawing a target around an arrow after it has been shot, rather than aiming at a target and seeing how close you get to it* [20]. He then proposes that the design phase should be broken down into the following steps:

1. Define product usability goals and objectives
2. Develop user scenarios and tasks
3. Define interface objects and actions
4. Determine object icons, views and visual representations
5. Design object and window menus
6. Refine visual designs

**Step 1: Define Product Usability Goals and Objectives**

Even at this point it’s tempting to analyze Mandel’s choice of words: *usable system*. In short, this is determined from a number of well defined easily measurable objectives that can be compared to a chosen objective or goal. Chapter 7 about software usability testing will return to this subject in detail. Nevertheless, Mandel’s words are closely related to the first step of this second phase. Design goals are indeed supposed to be measurable and are best expressed in terms as user behavior and performance, for example the time it takes to complete a task or how many errors are expected. The product usability goals and objectives are thus often broken down to the following four steps:

- Usefulness
- Effectiveness
- Learnability
- Attitude

These goals and objectives are then in turn validated for the two types of users defined in phase 1. The result of this analysis can be viewed in table 3. Naturally it’s possible to elaborate a great deal on this list and the usability testing chapter will return to this to a greater extent. The goals and objectives in this table are meant to serve as a basis to drive the design of the product interface.
<table>
<thead>
<tr>
<th>Product Goals and Objectives</th>
<th>Novice Users</th>
<th>Advanced Users</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Usefulness</strong></td>
<td><strong>Goal:</strong> Users will be able to use the program to perform their tasks</td>
<td><strong>Goal:</strong> Users will be able to use the program to perform their tasks</td>
</tr>
<tr>
<td></td>
<td><strong>Objective:</strong> 100% of the users will be able to use the program to create a</td>
<td><strong>Objective:</strong> 100% of the users will be able to use the program to create a</td>
</tr>
<tr>
<td></td>
<td>slideshow with the proper guidance (tutorial, wizard, teacher etc)</td>
<td>slideshow with minimal guidance (tutorial or wizard)</td>
</tr>
<tr>
<td><strong>Effectiveness</strong></td>
<td><strong>Goal:</strong> Users will be able to work faster than with current alternatives</td>
<td><strong>Goal:</strong> Users will be able to work faster than with current alternatives</td>
</tr>
<tr>
<td></td>
<td>Users won’t have to turn to the manual all the time because of the intuitive</td>
<td>Users won’t have to turn to the manual all the time because of the intuitive</td>
</tr>
<tr>
<td></td>
<td>interface</td>
<td>interface</td>
</tr>
<tr>
<td></td>
<td><strong>Objective:</strong> 100% of the users will be able to complete their task within</td>
<td><strong>Objective:</strong> 100% of the users will be able to complete their task within the</td>
</tr>
<tr>
<td></td>
<td>the given time constraints</td>
<td>given time constraints</td>
</tr>
<tr>
<td><strong>Learnability</strong></td>
<td><strong>Goal:</strong> Minimal user training will be needed</td>
<td><strong>Goal:</strong> Minimal user training will be needed</td>
</tr>
<tr>
<td></td>
<td><strong>Objective:</strong> Users will be able to use the product successfully after</td>
<td><strong>Objective:</strong> Users will be able to use the core functions in the product</td>
</tr>
<tr>
<td></td>
<td>doing a tutorial or with proper training</td>
<td>with no training</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Users will be able to use most functions successfully after doing a tutorial</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or with proper training</td>
</tr>
<tr>
<td><strong>Attitude</strong></td>
<td><strong>Goal:</strong> Users will be satisfied with the product</td>
<td><strong>Goal:</strong> Users will be satisfied with the product</td>
</tr>
<tr>
<td></td>
<td><strong>Objective:</strong> Users will rate their satisfaction with the product at a high</td>
<td><strong>Objective:</strong> Users will rate their satisfaction with the product at a high</td>
</tr>
<tr>
<td></td>
<td>level</td>
<td>level</td>
</tr>
</tbody>
</table>

*Table 3: Switch usability goals and objectives.*
Step 2: Develop User Scenarios and Tasks

A *user task* is considered a “small operation” that the user does in the program. An example can be “*opens file xxx*”. A *scenario* is typically a sequence of tasks that comprises a common transaction. It’s a high level description of what the user does or wants to get done. The scenarios will offer a base for the usability testing in chapter 7. In this chapter, the scenarios and tasks will be put into system, tested, and evaluated. Eight user scenarios and tasks are listed in table 4.

<table>
<thead>
<tr>
<th>Novice Users</th>
<th>Advanced Users</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1:</td>
<td>Scenario 1:</td>
</tr>
<tr>
<td><em>Basic knowledge.</em></td>
<td><em>Understanding different canvas elements.</em></td>
</tr>
<tr>
<td>The user wants to create a simple text-only slideshow with a custom title slide and no graphics or effects.</td>
<td>The user wants to create an image slideshow with a background picture and image and text animations:</td>
</tr>
<tr>
<td>User tasks:</td>
<td>User tasks:</td>
</tr>
<tr>
<td>• Opens the program</td>
<td>• Opens the program</td>
</tr>
<tr>
<td>• Chooses the slideshow background color</td>
<td>• Chooses the slideshow background image</td>
</tr>
<tr>
<td>• Inserts a textbox for the title slide</td>
<td>• Creates a textbox for the title slide</td>
</tr>
<tr>
<td>• Inserts the slideshow title in the textbox</td>
<td>• Edits the content and the font for the title slide</td>
</tr>
<tr>
<td>• Edits the textbox font properties</td>
<td>• Changes the textbox background color, border color and border width</td>
</tr>
<tr>
<td>• Changes the textbox background color, border color and border width</td>
<td>• Selects an animation for the textbox</td>
</tr>
<tr>
<td>• Inserts slide #2</td>
<td>• Inserts slide #2</td>
</tr>
<tr>
<td>• Inserts a textbox in slide 2, edits the text and changes the font properties</td>
<td>• Inserts an image box in slide #2</td>
</tr>
<tr>
<td>• Repeats for as many slides wanted</td>
<td>• Inserts an image in the image box</td>
</tr>
<tr>
<td>• Previews the slideshow internally</td>
<td>• Selects an image animation</td>
</tr>
<tr>
<td>• Saves the slideshow</td>
<td>• Repeats for as many slides wanted</td>
</tr>
</tbody>
</table>
Scenario II:

Understanding the slides list.
The user wants to open an existing slideshow with four slides, switch the order of slide 2 and 3, delete the last slide, and finally print the slideshow.

User tasks:
- Opens the program
- Opens the wanted file
- Rearranges slide 2 and 3 by either drag/drop, or by copy/paste.
- Deletes the last slide
- Prints the slideshow

Scenario II:

Understanding the menu box.
The user wants to create a slide in which several slide elements are linked to a menu box.

User tasks:
- Opens the program
- Creates a blank slide for the desired content
- Creates a few textboxes, image boxes or rectangles and ellipses that are going to be linked to the menu items
- Inserts a menu box
- Creates a few menu box items
- Links the graphical elements to the menu box

Scenario III:

Understanding background colors, background images, and inheritance.

Understanding the color dialog.

Working with the property grid.
The user wants to create four slides in a slideshow, give one of them a custom background color, one a custom background image, and the rest inherit from the slideshow settings.

User tasks:
- Opens the program
- Creates four slides
- Sets slideshow background color to a chosen color. Notices that all slides change background color.
- Sets slideshow background image to a chosen image. Notices that all slides change background images.
- Sets a specific slide color on one slide. Turns “Inherit from slideshow” off. Notices that the slide in edit gets a specific slide background color.
- Sets a specific slide background image on another slide. Turns “Inherit from slideshow” off. Notices that the slide in edit gets a specific slide background image.

Scenario III:

Understanding background colors, background images, and inheritance.

Understanding the color dialog.

Working with the property grid.
The user wants to create four slides in a slideshow, give one of them a custom background color, one a custom background image, and the rest inherit from the slideshow settings.

User tasks:
- Opens the program
- Creates four slides
- Sets slideshow background color to a chosen color. Notices that all slides change background color.
- Sets slideshow background image to a chosen image. Notices that all slides change background images.
- Sets a specific slide color on one slide. Turns “Inherit from slideshow” off. Notices that the slide in edit gets a specific slide background color.
- Sets a specific slide background image on another slide. Turns “Inherit from slideshow” off. Notices that the slide in edit gets a specific slide background image.
Scenario IV: Understanding the different file types.
The user wants to preview an existing slideshow in fullscreen mode using the Macromedia Flash Standalone player (or a browser), without opening the Switch program.

User tasks:
• Navigates to the file directory using a file browser such as Windows Explorer
• Chooses the flash file
• Opens the file in the Flash Player
• Switches the flash player to fullscreen mode

Scenario IV: Understanding the drawing tools and linking lines.
The user wants to draw a stickman in the slideshow.

User tasks:
• Opens the program
• Uses the drawing tools to create a desired stickman
• Previews the slideshow internally

| Table 4: User scenarios and tasks in Switch. |

Step 3: Define Interface Objects and Actions

Probably the most difficult and important step in the process is to identify the interface objects the users are going to interact with. Since one of the overall design goals in any interface design project is to hide as much of the complexity of the system as possible, many underlying objects and processes are hidden from the user [20]. For example when the user deletes a slide, a whole range of events will occur to ensure the file format is still intact, the ordering of the slides is updated, computer memory is preserved, undo operations can be completed etc. As far as the user is concerned, he or she only cares if the slide was deleted or not. At this point it’s essential to make a few strategic choices. First of all, it’s already been described that it’s rarely a good idea to deviate too far away from Microsoft de facto standards. The planned intuitiveness of the user interface is partly based upon the fact that the users need to recognize interface elements even if they have never used the program before. This is connected to the expectation of a user interface element, that is, what the user expects a toolbar button to do based upon his or her previous experience.

There are many ways to determine what the objects and actions in the interface are. Mandel proposes to develop a wide range of scenarios followed by underlining all verbs and nouns in the scenarios and user tasks [20]. While this might be a fairly good indication to user interface elements, it still doesn’t complete the list. The list needs to be completed through
brainstorming and evaluation. For the case of the Switch project, the toolbars play an important role. The buttons on the toolbars are traditionally considered as shortcuts to changes the user can do elsewhere, for example to a properties dialog box or a menu. What buttons to include is generally a matter of finding out what operations the user uses most frequently. Yet, the user interface consists of more than a toolbar.

Not all interface objects need a visual presentation. For example, there is no need to include an icon for the presentation itself. Even though the presentation has attributes the user might want to explore or set, these values are better accessed from a menu.

In this case, the most visible user interface elements will be the slides list, the slide canvas and the PropertyGrid. The slides list contains the list of slides in the presentation, the canvas contains a visual representation of the current slide elements, and the PropertyGrid contains the attribute-value pairs for the currently selected slideshow element. In addition to this, all the slide elements need a visual representation. A slide element is a textbox, image box, menu box, and the simple drawn elements such as a rectangle/square, circle/ellipse and lines and arrows. The elements and associated actions are listed in table 5.

<table>
<thead>
<tr>
<th>Interface Objects</th>
<th>Associated Actions</th>
<th>Misc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slides list</td>
<td>• Insert slide</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Delete slide(s)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Select slide(s)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Copy slide(s)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Paste slide(s)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Rearrange slides</td>
<td></td>
</tr>
<tr>
<td>Slideshow</td>
<td>• New</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Open</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Save</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Save as</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Delete</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Set dimensions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Set background color</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Set background image</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Set framerate</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Set comment</td>
<td></td>
</tr>
<tr>
<td>Slide</td>
<td>Slide canvas</td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>• Set background color</td>
<td>• Select textbox</td>
<td></td>
</tr>
<tr>
<td>• Set background image</td>
<td>• Select menu box</td>
<td></td>
</tr>
<tr>
<td>• Insert/delete textbox</td>
<td>• Select image box</td>
<td></td>
</tr>
<tr>
<td>• Insert/delete image box</td>
<td>• Select graphical objects</td>
<td></td>
</tr>
<tr>
<td>• Insert/delete menu box</td>
<td>• Insert/delete textbox</td>
<td></td>
</tr>
<tr>
<td>• Insert/delete graphical</td>
<td>• Insert/delete image box</td>
<td></td>
</tr>
<tr>
<td>objects</td>
<td>• Insert/delete menu box</td>
<td></td>
</tr>
<tr>
<td>• Insert/delete graphical</td>
<td>• Insert/delete graphical objects</td>
<td></td>
</tr>
<tr>
<td>objects</td>
<td>• Select multiple objects</td>
<td></td>
</tr>
<tr>
<td>• Select multiple objects</td>
<td>• Zoom</td>
<td></td>
</tr>
<tr>
<td>• Zoom</td>
<td>• Move/resize textboxes, image boxes, menu boxes and graphical objects</td>
<td></td>
</tr>
<tr>
<td>• Move/resize textboxes,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>image boxes, menu boxes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>and graphical objects</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Textbox</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Edit text</td>
<td>A textbox should have the possibility to contain fonts of different sizes, colors, bold, italic, and so on.</td>
</tr>
<tr>
<td>• Edit font</td>
<td></td>
</tr>
<tr>
<td>• Change size</td>
<td></td>
</tr>
<tr>
<td>• Change outline width</td>
<td></td>
</tr>
<tr>
<td>• Edit background color</td>
<td></td>
</tr>
<tr>
<td>• Edit foreground color</td>
<td></td>
</tr>
<tr>
<td>• Select animation type</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Image box</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Select picture</td>
<td></td>
</tr>
<tr>
<td>• Select animation type</td>
<td></td>
</tr>
<tr>
<td>• Change size</td>
<td></td>
</tr>
<tr>
<td>• Change outline width</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Menu box</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Change size</td>
<td>Contains menu items</td>
</tr>
<tr>
<td>• Change outline width</td>
<td></td>
</tr>
<tr>
<td>• Edit background color</td>
<td></td>
</tr>
<tr>
<td>• Edit foreground color</td>
<td></td>
</tr>
<tr>
<td>• Select animation type</td>
<td></td>
</tr>
<tr>
<td>• Insert menu item</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Menu item</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Edit text</td>
<td></td>
</tr>
<tr>
<td>• Edit font</td>
<td></td>
</tr>
<tr>
<td>• Link to slide objects</td>
<td></td>
</tr>
</tbody>
</table>
Table 5: Objects and actions in the Switch interface.

Mandel also proposes that the list should contain visual representations for elements such as a printer or a trashcan [20]. The printer will have a button on the toolbar, but the interface does not need a trashcan. The delete button and possibly a context menu choice will serve this purpose.

Step 4: Determine object icons, views and visual representations

The design of the icons themselves should as a general rule be left to a graphics designer [20]. However, Switch will not contain a set of icons for accessing objects and their interfaces. This is more related to web design.

The creator(s) of the program will often have some general idea of how and where to place the graphical objects and icons. Through brainstorming and prototyping, a few sketches
should be made to depict the outline of the interface. The Switch interface draft is shown in figure 12.

Figure 12: The first draft of the Switch interface.

The slides list is a representation of the slides in the presentation. To give a visual feedback of the slide contents, each slide in the slides list should be a miniature image of the slide as it looks on the canvas. The list is vertically aligned as in PowerPoint and placed on the left hand side of the interface, below the toolbars.

The slide canvas is shown in the middle. This is where the user modifies the slide contents. The user will add text through textboxes, images through image boxes, graphical objects through the drawing tools, and so on.

The PropertyGrid is shown to the far right in the figure. This is as explained a collection of categorized attribute-value pairs for the currently selected object. Each time the user clicks on something on the canvas or a slide, the PropertyGrid will automatically update itself with the properties for the selected object.

A textbox is a box containing text. The box can have an outline, a fill color, and the text can be of any font and color. The visual representation of the textbox must be as close to the rendered result in the final slideshow as possible. A textbox without text should be deleted by the interface. If the desired result is a rectangle, the rectangle drawing function is the proper way to do it.
The image box is quite similar to the textbox. It can also have an outline, but no fill color. Instead of text, the image box contains an image, scaled down to fit the size of the box.

A menu box is a special kind of box, new in the Switch interface. It is a vertically ordered list of menu items. Each menu item is linked to one or more elements in the current slide. When the menu item is clicked in the rendered slideshow (that is, not in the interface) the element that is linked to this menu item becomes visible. The menu items should be rendered as text inside a bounding box. Linking objects is performed by dragging a link handle from the menu item to the item that is to be linked. This relationship should then be visualized using a line between the two objects when the mouse hovers over one of them. The user should also have the possibility to display all links at once.

A rectangle has an outline or/and a fill. The outline has a line width and a color. The fill is a solid color or a linear or radial gradient of two colors. At this point it’s important to emphasize that Switch is not a drawing program. There are far better tools out there that serve this purpose. Yet, a few basic drawing functions are to be implemented.

The ellipse is very similar to the rectangle. They have all the same fill and coloring properties, and are separated by their shapes.

A line has a width and a color. The rendering on the screen is exactly that. In addition to just drawing lines, Switch is going to offer the possibility to connect lines. This will be shown by drawing a small white box on top of the joint between the lines. The user should be prompted whether he or she wishes to connect the lines or not.

On top of this, the interface needs a way to distinguish between a selected element, several selected elements together, and non-selected elements. Examples of selected elements in three different programs are depicted in figure 13, 14 and 15.
In Switch, there is a distinction between what kind of element is selected. The slides list should mark the current slide by graying the background of the list “cell” the slide resides in. There could also be a distinction in the slide size such as marking the currently selected slide as bigger than the rest in the list. This approach is greatly inspired by the Dock on the Apple
OSX operating system (figure 16). An implementation of the slides list similar to the Dock saves a lot of space since only the chosen slide is visible, yet it should be optional to accommodate less experienced users.

Figure 16: The Apple OSX Dock enlarges the icon the mouse hovers over, and gradually reduces the sizes of the nearby icons.

The slide canvas elements must use a different approach. As already mentioned, different programs use different approaches to visualize the currently selected element. PowerPoint, for example, has for some reason chosen to visualize selected elements in different ways (figure 17).

Figure 17: A selected circle, textbox and line in PowerPoint.

At first glance this does not seem like a coherent way to display selected objects. Obviously, a line should not be outlined with a bounding box since the bounding box will simply take up too much space and disturb the overall impression, but the rest of the objects should be outlined in a common manner. Switch will take the PowerPoint textbox approach and outline
interface elements with a hatched bounding box with eight grab handles. This is illustrated in figure 43 on page 88. A selected line will get a hatched color and two grab handles.

**Step 5: Design object and window menus**

The Microsoft Windows guidelines offer no suggestions as to when menu bars are appropriate. IBM, however, state in their guidelines that a menu bar should be visible “when a window will provide more than six action choices or routing choices” [21]. Switch clearly has more than six action choices. Furthermore, a design goal is to not deviate too far away from Microsoft standards. The obvious choice is to let the main interface have a menu bar. When it comes to dialog boxes, they seldom have menu bars. Some applications use multiple windows and hence need more than one menu bar, but Switch is not one of those. The main interface menu bar is therefore the only one.

Microsoft Windows Guidelines do state that menu choices should be reduced at the expense of dialog boxes. They also state that the holistic user experience is improved by having consistency across programs rather than forcing the user to use a whole range of products where all products have widely varying interfaces. A user should quickly be able to recognize common interface menus and elements and they are usually similar across a wide range of programs [22]. The Switch menu should be no exception. Table 6 lists the Switch menu hierarchy after the first paper prototyping iterations. After the program gained complexity, the menus themselves served as a “living spec” for the menu system, similar to what was done during the Windows 95 development process [38].

---

6 A grab handle is a circle or a box on an element’s bounding box, placed in the four corners and on the midpoints of all four sides of the bounding box. The grab handles are usually used for resizing or rotation of the element.
<table>
<thead>
<tr>
<th>Menu bar</th>
<th>Submenu level 1</th>
<th>Submenu level 2 / misc.</th>
</tr>
</thead>
</table>
| **File** | • New Presentation…
|          | • Open Presentation…
|          | • Close
|          | • Save
|          | • Save As…
|          | • Print…
|          | • Print Preview…
|          | • Page Setup…
|          | • Publish…
|          | • Publish Settings…
|          | • Exit |
| **Edit** | • Undo ➔ and name of undo action
|          | • Redo ➔ and name of redo action
|          | • Cut
|          | • Copy
|          | • Paste
|          | • Delete
|          | • Select All
|          | • Find…
|          | • Settings… |
| **Insert** | • New Slide…
|            | • Picture ➔ From Library…
|            | • From File… |
| **Format** | • Font…
|            | • Slide…
|            | • Presentation… |
| **Preview** | • Preview Current Slide…
|            | • Preview Presentation… |
| **Windows** | • Presentation # 1
|             | • Presentation # 2
etc… |
| **Help** | • Contents…
|            | • Index…
|            | • Search…
|            | • About Switch… |

*Table 6: The Switch menu hierarchy.*
There is also need for a few context menus\(^7\). The slides list needs a context menu for slide management. It should have the following choices:

- Insert Slide Before
- Insert Slide After
- Cut
- Copy
- Paste
- Delete
- Slide Properties…

The slide canvas objects should have a similar context menu:

- Cut
- Copy
- Paste
- Delete
- Unlink (if applicable, e.g. for lines grouped together)
- Explode (if applicable, e.g. for lines grouped together)
- Properties…

The links between a menu item and an item on the slide canvas should have a context menu with simply:

- Unlink

…and finally, the “joints” that join the lines in a figure with lines grouped together should have a context menu with:

- Open this joint
- Open all joints in figure (explode)

**Step 6: Refine visual designs**

The last step in the second phase is to get an overview over the analysis up to this stage. It’s often easier and faster to create the interface on paper before programming it. Typical

\(^7\) A context menu is a menu accessed by right-clicking the object.
inconsistencies can be spotted and removed before they impact the implementation process. In this project, numerous minor inadequacies have been corrected over several iterations in an ongoing refinement process. All different parts of the interface were subject to paper prototyping and informal testing before the actual coding began.

4.3 Phase 3: Construct the User Interface

The first iterations of any bigger design process are almost always performed through prototyping. Prototyping provides a valuable lightweight version of the user interface and is necessary for early usability testing. A programmer or a team should not be afraid to throw away elements in the user interface, maybe even the whole prototype, and the purpose of prototyping is to quickly and easily visualize design alternative ideas. The code must not necessarily be a part of the final product.

Prototypes come in many forms. They vary from the handwritten early sketches with a pen and pencil to advanced computer animations. Prototypes can show how specific transactions can happen inside a product or demonstrate high-level concepts. There are advantages and disadvantages to all forms.

Today, many application development environments are so called RAD-tools\(^8\). These tools can make the design process a lot easier and make the envisioning of the user interface elements much better. Many tools also generate code for the final product. Switch is written in the C# programming language using Microsoft Visual Studio. Visual Studio is also a programming environment so it will act as both development tool and prototyping tool.

Many huge development projects follow a “prototype as specification” approach. When Windows 95 was developed, after the first few months the design spec had grown very large and represented hundreds of person hours of work. The problem was that due to problems uncovered after usability testing, the document was not up to date with the program code. The development team faced a major decision: Spending weeks updating the design specification and loose valuable time that should be used for iterating, or let the prototypes

\(^8\) RAD is an acronym for Rapid Application Development, an environment that lets the programmer or designer drag and drop buttons and other interface elements directly onto the program canvas instead of programming the locations by hand.
and code itself serve as a “living spec”. After some thought, the latter approach was chosen [23].

For the detailed discussion of the development of the Switch user interface, see chapter 6, *The Design of the Switch User Interface*.

### 4.4 Phase 4: Validate the User Interface

Usability testing is, and should be a major part of any development project. It’s a way to put the product in the hands of the users to see whether they can use it or break it. Unfortunately, many development projects don’t address this issue until the very last stage of the development [20]. It’s also bad practice to behave as if the development process is linear and only test once. Just think about it, how can you be sure if the changes that were made to an interface were the right ones without testing again? The usability test should to some extent be visual through all phases of the development process. From the early beginning through concept definition, concept validation, design, development, pilot versions, beta tests and so on. Many programs today even continue the evaluation and feedback from deployed programs after they’ve been sold to the customer [20]. From the information they collect from error handling, user habits and feedback, the next version of the application can be made even better. This borders to the concept of data mining, which is not the subject of this thesis.

Also the developers should observe the users when they test the program. That way they can get a feel for how the users interact with the program and provide technical assistance if necessary. The users should have various levels of expertise, and they should not be a member of the development team. To let the developers test their own products just doesn’t work [20].

Usability testing is an art. In the real world, it requires lots of time, money and effort. No two cases are the same. During the usability testing phase, the scenarios developed under phase two in this chapter will come to use again. The product must be able to function in the given scenarios. Switch will undergo these usability tests and evaluated thereafter. This testing phase is described in detail in chapter 7, Software Usability Testing.
5. Designing a Good User Interface

Despite its importance, human-computer interface experts claim that the human-computer interface is one of the most poorly understood aspects of interactive systems [35]. Baecker et al. note: “The success or failure of an interface is determined by complex, subtly interrelated issues. For example, is it responsive or sluggish, forgiving or intolerant of human error, easy or difficult to learn, easy or difficult to use? Even whether or not it’s attractive and fun to use can make a significant difference.”

5.1 Being in Control

The design of a good user interface is in many respects imperative to the success of the application. This is because the user interface affects the feelings, the emotions and the mood of the users [38]. A user that has problems using an ill-designed user interface will probably stop using the application altogether and look for alternatives. If the user has to use the application (because of inexistence of alternatives, job situation etc), he or she will probably feel resentment towards the application because it is simply not possible to control the program environment the way the user wants to. On the other hand, applications with a good user interface may impact the user in a positive way. The user is in control, receives the proper feedback, and the application behaves the way it should (assuming, of course, that the rest of the application is programmed properly).

The above observations are rooted in an important theory of psychology called learned helplessness. This theory was developed by psychologist Dr. Martin E. P. Seligman in 1965 [37] and is backed up by years of research. In short, the theory is that a great deal of depression grows out of a feeling of helplessness: the feeling that you cannot control your environment or that the environment controls you. The more a user can feel that he or she is in control over the environment, and that the things the user does are actually working, the happier the user is. When a user feels anger or frustration, it’s often because something happened that the user was not in control over. It can be a small thing. It can be many small things. The clue is often that a lot of things happen and they add up. Add enough of these situations and the user becomes relent [37].
Similarly, consider for a second an intuitive environment. The user might not be entirely sure of how to use all functions in a program, but usually may have some general idea of what he or she wants to achieve. With an intuitive environment, the desired goals are should not be that difficult to accomplish. This way the user experiences a lot of tiny victories and they add up. The result is a happy user.

This brings us to what Joe Spolsky calls the cardinal axiom of all user interface design [38]:

*A user interface is well-designed when the program behaves exactly how the user thought it would.*

### 5.2 Conceptual Models

When a user first tries to use an application, he or she does have some general understanding of how computer programs work. Chances are they’ve seen a toolbar, perhaps the Internet Explorer window, knows what a button is, and so on. The point is that the users don’t start from scratch with each new program they try. It might help to imagine the users as people with suitcases (figure 18).

![Figure 18: A model of people, as they always bring with them some kind of "mental baggage" when they solve problems.](image)

The model of people and suitcases is an important metaphor for how people think when they first encounter a problem or are trying to find a solution to something. The suitcase is their “mental luggage”, a symbol for knowledge they have acquired using other software or from solving similar problems in general. No two people’s suitcases are the same, all users are different. Some come with a pretty full suitcase, others’ are quite empty.

When the users try to use an application for the first time they have some expectations of how the software is going to work. If a user has had experience with similar software, he or
she will assume the program will work as the other application. In fact, if the users have
tried any software before at all, chances are that they will believe the software conforms to
certain conventions. They will probably also have intelligent guesses about how the user
interface works. This is the user model: it is the users’ mental understanding about what the
application is doing for them[38]. Mental models are important concepts in design. The
users have mental models of themselves, others, the environment and the things with which
they interact. A model a user has of an artifact will largely impact the way the user interacts
with it. When people form models, they usually do it through experience, training and
instruction. Mental models of objects are thus a result formed by users interpreting the
object’s perceived actions and visible structures[91].

Spolsky introduces something he calls the program model[38]. The difference is that this
model is encoded in bits and bytes, and is executed faithfully by the CPU. This is the
program itself. When the program model and the user model correspond, a well-built user
interface has been created[38].

This introduces the programmer’s model and the designer’s model. The jobs of the
programmer and the designer (not necessarily two people) are to hide the underlying
complexity of i.e. computer hardware, objects and database transactions. They thus want to
map the interface to match the user’s mental model as closely as possible. The designer’s
model is hence conceptual model of the system, whereas the programmer will create his
model based on e.g. the system capacities, restrictions and the programming language. The
gap between the programmer’s world and the user’s environment is bridged by the
designer’s model[90][91].

As an example, a parallel to all this can be drawn to the development of the Switch file
format. When the user wants to embed an image in a presentation, the user includes an
image box. The image box should provide the abstraction for “image in presentation”. Once
the image is embedded in the slideshow, the user most likely happily assumes that the image
is a part of the presentation and separated from the original file. The problem is that all
images used this way in such a program technically are stored full-size in memory as a

[38] The user model is sometimes referred to as the mental model, the user’s mental model, or the user’s conceptual model [90]
bitmap, regardless of whether they originally were a jpeg, gif, png, or another compressed file format. Add a few of these images and they quickly consume vast amounts of memory. So, if the images can’t be stored in computer RAM, they would have to be stored somewhere else. A quick fix would thus be to store the images on disk, and instead store the paths to the images in the file format - just like html. Now, as trivial as this may seem, it probably conflicts with the user model. The reason for this is because the user probably assumes the image now is a part of the slideshow. Furthermore, the user might want to delete the image after it has been embedded in the presentation. After all, the sole purpose of that particular image might have been its usage in the presentation, and now that it’s embedded there is no use for it anymore. The result can be that the user deletes the image, and it’s lost both on disk and in the presentation. Chances are slim the user model included “references to image files” when the users embed images.

At this point, the developer has two choices. The first choice is to change the user model, but this turns out to be remarkably difficult. For example, a section in the manual could be devoted to explaining the users why embedding images simply results in a reference to the file on disk and that it cannot be deleted if the presentation is going to function properly. Unfortunately, users don’t read manuals. Not unless they really have to.

Another approach could be to pop up a small dialog box that explains the situation. This, however, actually presents two problems. First of all it’s annoying for advanced users because it gets in the way of everything else they are doing. Secondly, users normally don’t read message boxes either. Thus, if the user model cannot be changed, the only feasible alternative is to change the program model.

In Switch the problem was solved by embedding the images in the proprietary file format, byte-for-byte, one after the other. This way the references to the beginning of each image (and the size of the image in bytes) were programmatically stored as a pointer to a bytewise offset in this file. All this metainformation is then saved in an index table in the beginning of the file. This way the program can extract a chunk of bytes the size of the image from the

---

²As a side note, Microsoft PowerPoint actually does store references to embedded video files as the default setting, and doesn’t embed the video file itself. This has probably led to many a frustrated presenter as they might have copied their slideshow (and only the slideshow) to a CD, but during the presentation only a “broken link” symbol appears where there was supposed to be a video.
bytewise location indicated by the pointer and the original image is totally excluded from the file (figure 19).

**Figure 19: Diagram showing how images are embedded in Switch. The small rectangles below the images and the slideshow represent the files (the actual bits and bytes). In the slideshow file the images become part of the slideshow file itself, and are independent of the original images.**

There is a simple solution to finding out what the user model is – just ask the users! When the usability testing is being conducted, asking the users to think out loud about what they do and why they do it will usually return the answers needed. How to conduct usability tests and the number of test participants needed is the subject of chapter 7. The general rule, though, is to keep the user models simple. When people sit down with a program and have to guess how a program works, they tend to guess simple things [38]. It’s hard enough to make the program model conform to the user model when the models are simple.

### 5.3 Design Methodologies

When designing an application, it’s important to reflect over the design process. To arrive at the best possible solution, the right methodology must be chosen. Otherwise one might end up with a program that works fantastically, but that is significantly wrong for the target user groups.
In the past the design of software and interfaces was totally driven by the current technologies and the systems on which the software was built [94]. This was called system-driven or technology-driven design. Users were not part of the development process at all.

Since the early 1980’s the focus has shifted to user-centered design. According to Norman, a user-centered design philosophy is based on the needs and interests of the users, with an emphasis on making the products usable and understandable. The design goals are quite clear and among other things involve making the conceptual model of the system visible, making it easy to determine the actions possible at any given time, and making it easy to evaluate the current state of the system [93]. Although the users are supposed to be included in the design process, it’s usually in a passive role [94].

Now, however, the HCI community partly seems to move in the direction of user-involved and learner-centered design processes. The former involves looking at the users not simply as objects of study, but as active agents within the design process itself. This very similar to user-driven design, where some of the users become actual or de facto members of the development team [95].

As for certain development processes, there is also a need to use a task-centered design process. Such a process easily coincides with and elaborates on the methodologies above. The process involves focusing on the tasks the different users are going to perform and keep that in the center of the design phase [96]. By focusing on the actual tasks the users commonly perform, the user interface can adapt and simplify the process of doing these tasks. And example of this is the creation of a new slide or slideshow in Switch. Whenever a slideshow is created, the program automatically inserts the first slide. After all, there is no point in the user doing that every time. Furthermore, when a slide is inserted, it is always given some initial content. That content is usually two textboxes, but one can imagine the system in the future as making an intelligent choice of content based on the current content.

Microsoft has taken this a step further and implemented activity based planning in many of their products. This involves having the system asking the users what activity they want to do, i.e. create a birthday card. Many users find that this approach makes a system a lot easier to use since the system will lead them through the steps to complete the activity [38].
5.4 Making Choices

A common understanding in GUI development is the conception that a user usually does not want to use the application [39] [90]. Users want to get the job done, preferably as quick and easy as possible. The software is a tool aiding that purpose and the more it can be kept out of the way, the better. Effort spent on using the application is effort not spent on the work the user is trying to do. To emphasize, the users do care about the task they want to accomplish. If the users are using a graphics program, they probably want to have control over every single pixel down to the finest level of detail. Again, the program is helping them to achieve what they want to do.

When a development team is creating a piece of software, they have to make decisions for the user. In fact, design has been said to be the art of making choices [38]. Some constellations of choices would for instance not make sense, and it’s the development team’s responsibility that the user doesn’t have the possibility to make erroneous choices.

A project development process usually includes several developers and sometimes multiple teams of developers. Having more than one authority deciding how to present these decisions in the interface can sometimes lead to the development teams having internal disagreements on how to present the user interface and the user options. One branch might claim setting A is the best and another will argue that setting B is better. A not uncommon solution to this problem is to provide both settings A and B as options, and let the user decide. This approach, deciding not to decide, might not necessarily be a bad thing (although it usually is), but it has its implications. First of all, some choices don’t really interest the user. In fact, users care about a lot less than what one might think and it’s unnecessary to inflict unimportant choices on the users simply because the designers couldn’t decide which option is the better [38]. An example of this is the Find Setup Wizard (figure 20). Joe Spolsky calls this “...the most moronic “wizard” dialog in the history of the Windows operating system. This dialog is so stupid it deserves some kind of award” [38]. First of all the dialog is distracting. The user is trying to find the help file, but is instead presented with a wizard. Second of all it’s confusing. It offers to create a list (or database), as if this is interesting to the user (who is merely trying to find help), and continues to mention help file(s). Now the user is probably even more confused because there might be more than one help file(s). Last, most people don’t understand what the wizard is really asking. Even
people with PhDs in databases and indexing services will probably have a hard time understanding what the implications of choosing from this dialog really is. The irony of it all is that the programmers obviously had some idea of what was the ideal choice. They did after all go through the trouble of recommending one of the alternatives.

Another example of unnecessary options might be the Microsoft Word menu bar. For some reason the development team decided to implement this bar as dragable, likely to more frustration than benefit to the users. The main reason why this feature is frustrating is that the little ribbon that lets you drag the whole thing out of place is placed only a few pixels to the left and to the top of the most common GUI elements, namely the “File” menu and the “New file…” button. This makes it an easy glitch of the mouse to undock the whole menu bar to the frustration of the user. A not totally uncommon situation is hence shown in figure 21, where the menu bar is located in the one place nobody wants it to be.

Figure 20: A confusing and distracting wizard.
There are more reasons why the designers and programmers should consider thoroughly what options and decisions to pass on to the user. A good point is that users should not be forced to make decisions simply because many users become increasingly uncomfortable if they are forced to choose between options they don’t understand. It is therefore generally important to minimize the number of decisions that people have to make, and this brings us to the second cardinal rule of user interface design [38]:

*Every time you provide an option, you’re asking the user to make a decision.*

By all means, users need to have the possibility to make some decisions. For example, users tend to love to change the visual look of things to personalize the interface. In these areas the developers can go wild, but must remember to simplify the user decision-making by removing unnecessary choices. There will still be plenty of ways for the users to customize their documents, and for anything that integral to the work of the user adding plenty of options is usually a good thing. The important part here is for the development team to make the right decisions on which options the user should have the opportunity to choose from.
As an amending anecdote to this section it’s worth to mention that users don’t read. They don’t read the manual unless they have to and several studies show that the different users don’t read option boxes, message boxes or wizards [38]. In fact, the more words a programmer put in a dialog box, the less likely it is that the user will read it. Usability tests also show that [38]:

- Advanced users skip instructions. They assume they know things already and don’t have time to read complicated instructions.
- Most novice users skip the instructions. They don’t like reading too much and hope the default settings will be ok.
- The remaining novice users who do read the instructions get confused. The sheer number of words and concepts are confusing and this leaves them insecure.

If the requirements do call for some kind of wizard or options dialog, the programmers should shorten it, simplify it as far as possible, and naturally usability test it. This reduces the burden on human short term memory. Research namely shows that the average person can retain only 7 ± 2 “chunks” of information in short-term memory, and that sentences with more than 25 words are perceived as overwhelming [41]. This principle is stressed in numerous articles and books [90] [50].

To address the question of reading the manual, the designers should design the software so that it doesn’t need the manual in the first place.

5.5 Mouse Control

Many users have problems controlling the mouse. This can result from a number of things, and don’t necessarily mean they lack the motor skills to use the mouse precisely. A program should hence be developed so it doesn’t call for a large amount of mouse-agility to use it right. Joe Spolsky lists the following examples [38]:

- People might be using sub-optimal pointing devices like trackballs, trackpads, or touchpads.
- Sometimes people use mice under bad conditions like a crowded desk, or the mouseball is dirty and skips.
• Some people are new to computers and haven’t developed the motor skills to use the mouse precisely.

• Some people might never develop the motor skills to control the mouse accurately. They might suffer from arthritis, tremors, carpal tunnel, or have any number of disabilities. They might also be very young or very old.

• Many people for some reason find it very difficult to double click without slightly moving the mouse. As a result, they often mistakenly drag things around when they meant to double click something.

• Using the mouse feels slow to a lot of people. If the users are forced to perform multi-step operations using the mouse, they may feel they are being slowed down. This in turn makes the user interface feel unresponsive.

Also *The Windows Interface Guidelines for Software Design* discourages the usage of the mouse in some areas. As it turns out, many novice users have problems understanding mouse-drag and double click operations, and they therefore encourage designers to create systems that do not rely on these operations [40].

When it comes to mouse control, Fitts’ Law is provides insight. It is one of the most basic and well-known of all user interface design laws [39]. In ergonomics, Fitts' law is a model of human movement, and it predicts “the time required to rapidly move from a starting position to a final target area, as a function of the distance to the target and the size of the target” [42]. To GUIs the law translates into something like “the larger and nearer an object is to the mouse pointer, the easier it is for the user to click it”. It might sound trivial, but is often completely ignored in user interface design [39]. In GUIs, Fitts’ Law can be easily put into practice. By i.e. finding out which buttons in the interface that are more frequently clicked than others, Fitts’ Law implies that these buttons could justifiably be made bigger. In figure 22, this principle can be seen in practice. Microsoft Internet Explorer gets it right by making the Back-button bigger than the other buttons. Clearly, this is the most frequently used button. Firefox on the other hand has chosen to make the Back-button the same size as the other buttons, making it more difficult to hit with the mouse.

Examples of the use of Fitts’ Law in Switch are the big foreground and background color select buttons on the toolbar, shown in figure 40 on page 85.
A principle that is to a large degree violated in all Windows operating systems up till today is a principle well-known from the Macintosh-world. It is based on the concept of the five most easily clicked areas on the screen: the four corners and the current position of the mouse [39]. A user interface evangelist and founder of Apple’s Human Interface Group named Bruce Tognazzini invented the concept of “the mile high menu bar” to explain why the menu bar on Macintosh machines is a lot easier to use than the Windows menus. The Macintosh menus are “glued” to the top of the screen and are not included in the application windows themselves, like in Windows. To hit the menu, all the user has to do is to slam the mouse pointer to the top of the screen. To hit the most commonly used menu, usually the File-menu, the user simply throws the mouse up and left and clicks. In Windows on the other hand, the user has to target a small area just off the border of the application window.

The principle is furthermore violated when it comes to scrollbars. In a maximized window, the scrollbar could have been located to the far right on the application window, but is instead located a few pixels to the left of the screen edge. Even these few pixels make a huge difference and it makes targeting the scrollbar so much more difficult. Unfortunately, recreating window look and feel can in turn violate the principle of consistency, which will be discussed later. Firefox, however, seems to have found a solution that tackles this problem and still lets the scrollbar be “mile wide” (figure 23).
In this example Firefox gets it right by enabling the scrollbar to be handled even at the far right pixels of the window. Internet Explorer has a gap only a few pixels wide.

From the above discussion one can thus deduct that the controls that should be easy to hit are best placed at the edges or the corners of the screen.

5.6 The Elements of a User Interface

Before continuing, it’s about time to take a brief look at the elements that comprise a common graphical user interface. The controls that are presented here are by no means an exhaustive list, but intended as a short description of the most frequent controls used in the user interface of the Switch project.

To understand what a control really is, a little basic understanding of the Windows control hierarchy is necessary (figure 24).
Figure 24: Windows forms component hierarchy.

Taken directly from Microsoft’s MSDN³ webpages, a control is defined as “...a child window an application uses in conjunction with another window to perform simple input and output (I/O) tasks.” [43]. In plain English, a control can be viewed as a piece of “screen real-estate” that contains a number of functions, behaviors and properties that are common for

³ Microsoft Developers Network.
the controls used in a live application. Buttons, scrollbars, tab pages, labels, and toolbars are all examples of controls that are commonly used in applications. They all ultimately derive from the same base class, which is the Control class. On its own, a control doesn’t do much. The magic happens when the control interacts with the Windows Forms engine. The operating system will then instruct the control to paint itself, or to execute whatever additional code the programmer might have coded into the control as a response to e.g. events. It’s worth to notice that a control contains a collection of other controls (whether the collection is in use or not), so-called child controls.

**Form**

One of the most common container controls is a form. It is what users usually refer to as a “window” and is depicted in figure 25.

![Figure 25: A form with a label and a button](image)

A form can, and usually will, host a range of child controls. These can be buttons, list boxes\(^4\) and so on, but also other forms. As already mentioned, the Form class derives from the Control class, which can hold other controls. Normally, if a form is hosting another form, the purpose is to provide an MDI interface. MDI is short for *Multiple Document Interface*, and is shown in figure 26. Earlier versions of Microsoft Word and Excel used to be MDI applications, but have now been changed to *Single Document Interfaces (SDI)*. Microsoft officially states that SDIs are easier to use and preferred [44]. Switch is at this point an MDI application.

---

\(^4\) According to Microsoft guidelines, referring to controls in documentation is done using lowercase whitespace-separated words. When referring to the class itself, such as ListBox, the class name can be used and it is capitalized [45].
Figure 26: MDI interface: a parent form and two child forms.

**Label**

A label with the text “Hello world!!!” is depicted in the form in figure 25. The purpose of a label is to provide non-editable non-selectable textual information at a given location on a form. Even though many controls provide textual information, labels are a nice way to offer short, concise information or non-critical feedback. Changing the text of a label to inform of critical user actions is usually not good because it can easily be overlooked by the user.

**Button**

Also a button is depicted in the form in figure 25. Buttons are a great way for the user interface “to make things happen”. They also boast great *affordance*, a concept that will be discussed in detail in the next section.

**Group Box and Panel**

The probably most common container controls are the group box and the panel. They are both used to serve as a “canvas” where other controls and operations take place. The panel is more or less invisible. Unless the programmer specifically gives it a color or a border, it will inherit the color from the parent control and be displayed borderless.
The group box, however, does have both a border and a label. It is mainly used to group elements of similar behavior together, such as options and preferences (figure 27).

![Example](image)

*Figure 27: A form with a group box containing a check box and two radio buttons.*

Windows guidelines state that group boxes should contain nothing but controls with a clear relationship and that group boxes should not be nested. All group boxes should be labeled and the labels should be short and concise [45].

### Check Box and Radio Button

Check boxes and radio buttons are strongly related to buttons. They are both displayed in figure 27. Check boxes provide a means for the user to accept different values where zero or more of the given values are allowed to be selected at the same time. Radio buttons are similar. By using a radio button, the user can choose between related values where exactly one of the values must be provided.

### Textbox

Textboxes are commonly known in two forms, single line and multi line (although they are really both the same textbox with different properties set). They provide a means for the user

---

5 A container control is a control whose main purpose is to host other controls.
to input textual information or for the application to display more complex information that is too large for a label (figure 28).

![Example](image.png)

*Figure 28: A form displaying two textboxes and a picture box (and two labels).*

In the figure above, the file information textbox has been set to read-only mode and multi line.

**Picture Box**

Also a picture box is displayed in figure 28. It is one of the simplest controls in the .Net framework and displays an image. In addition to this, the control offers to automatically resize the image to fit the size of the control, or to crop it.

**List Controls**

.Net provides three basic list controls: List box, combo box, and checked list box (figure 29).
Figure 29: Three different list controls.

Also these controls are related and they are meant to display data from e.g. a dataset or an array. The controls are fairly complex. Therefore, to go into an in-depth description of these controls is outside the scope of this thesis. Furthermore, the checked list box is not used in the project, but nevertheless a part of the .Net list controls and hence displayed here. The slides list in Switch is an owner drawn list box and will be described further in chapter 6.

Custom Controls

The controls provided by the .Net framework are often not versatile enough to provide all the functionality required by an application. Sometimes the controls need images, animation, custom looks, or special properties, events or functions. Occasionally controls might even need a non-rectangular shape or transparency. This is where custom controls come in. They help the programmer by providing encapsulation, a simpler programming model and it makes the user interface more flexible as it is easier swapping one control with another. Custom controls naturally enhance the user interface by providing custom functionality.

Developers usually make a distinction between three types of custom controls\(^6\) [77]:

- **User controls**: These are the simplest type of custom controls. The most common use of user controls is to provide some kind of logic and programmatic grouping of controls. An example of a user control is the popup panel for resizing slides in the

\(^6\) Some developers also consider *extender providers* a fourth form of custom controls, although these aren’t really controls at all. These components add features to other controls on a form.
Switch user interface (bottom left corner of figure 35). Here, one or two custom track bars are used to adjust the slide sizes in the slides list, with an accompanying floating indicator that displays the slide size.

- **Inherited controls**: These controls are generally perceived as more powerful and flexible. The general recipe is to pick the already existing control with the closest behavior to the wanted control, and then derive a custom class from this control. In the custom control the programmer can add properties and methods, and override existing behavior. An example of an inherited control is the color buttons on the Switch toolbar (figure 40). They behave and look more or less exactly as buttons, but display a thin white frame just inside the control border.

- **Owner-drawn controls**: The most powerful (and thus most complicated) of the custom controls are the owner-drawn controls. These controls are often derived from the Control base class and programmers implement their interfaces pretty much from scratch. They provide the most customizable interface and generally use GDI+\(^7\) drawing routines to generate their interfaces. An example of such a control is the selection frame of the graphical objects in the Switch interface (figure 43). This is actually a control shaped as a rectangle with a hole in it.

### 5.7 Psychology Behind an Intuitive User Interface

An intuitive graphical user interface is an interface that makes it clear how the underlying program functions [4]. Naturally, users are different and intuition is a very individual term for each user. However, there are many ways to target intuition by using references to elements people will recognize. Workers in the fields of perceptual and cognitive psychology, as well as human-computer interaction researchers have asked themselves “What is it about this object that makes people want to use it this way?” The objects around us seem to talk to us with some sort of language and this can be translated to computer

---

\(^7\) GDI+ is the evolution of GDI, which is an acronym for Graphic Device Interface. It serves as a simple layer that enables programmers to manipulate graphics without having knowledge of the underlying hardware technology.
interfaces. If industrial designers can create objects that explain their own functions, so must be the case also for computer interfaces.

Metaphors

Perhaps the most famous metaphor in the world of computers is the desktop [38]. On the desktop people have little folders and they can move files by dragging them from one folder to another. In the real world, the desktop is supposed to remind people of the top of a desk and the desktop metaphor works because the small folders actually do look like folders that make people realize people can put documents in them.

Another example of a good desktop metaphor is the trashcan. When the users want to dispose of their files, that’s where they put them. The concept of the trashcan also goes a little further. As with a real-world trashcan, users are also allowed to retrieve recently disposed items from the trashcan, just like it can be done in real life. On top of it all, not only does the computer desktop behave like a trashcan, it also looks like one and looks different when it is empty from when it is full (figure 30).

Figure 30: Metaphors in practice. This trashcan from the Macintosh environment has different looks when it is full and when it is empty.

Metaphors are an important part of any intuitive user interface. They allow the users to transfer their knowledge about how things should look and work onto the user interface element [65]. Nonetheless, the metaphors need to be chosen wisely or else the metaphor might not serve its purpose or even work against it. Naturally, layout and gestalt principles must still be followed.

As intuitive as the above trashcan examples may seem, the reader might find it amusing that the above metaphor hasn’t always been overly successful in all areas intended. For example, in ye olden days, disks didn’t work as flawlessly as they do today. Floppy disks were on several operating systems supposed to be unmounted to make sure the disks didn’t get
damaged. In fact, on old Mac machines the floppy disk drives didn’t even have an eject button. A developer at Macintosh got the idea that the trashcan could hence serve two purposes. Besides from throwing away stuff, the users had to throw the icon of their floppy disk into the trashcan to get their floppy disks ejected! As obvious as it might have seemed for the development team, the general idea was now that sometimes you put things in the trashcan when you don’t want to keep it, and sometimes you throw things in the trashcan to make sure they don’t get damaged [47].

The Macintosh trashcan metaphor usability also has another side to it. As a way to demonstrate that the bin had items in it, the development team decided to create two icons. One was picturing an empty trashcan and the other a full one (figure 31).

![Figure 31: The not so successful tweaked Macintosh trashcan.](image)

This might have been initially a good idea, but in practice it didn’t work that well. As soon as the users saw the trashcan swell up, they developed a bad habit of emptying the trashcan as soon as they put things in it [47]. From the above one can thus conclude that it is important to choose the metaphors well.

A few metaphors that do work well can be found on the toolbars of many common applications (figure 32). The open (…folder), print, and clipboard icons are good metaphors that spawn recognition. The print preview button, however, is a more questionable metaphor as it looks more like a magnifier. As a side note one might wonder whether the save button depicting a floppy disk will have the same metaphorical effect on the next generation computer users.

![Figure 32: Common toolbar metaphors.](image)
Affordances

In industrial design, well designed objects make it clear how they work by just looking at them. Some doors, for example, have a big metal plate instead of handles and it’s obvious that to open the door, you have to push on the metal plate. The plate affords pushing. Other doors have big handles on them that almost invite you to pull the knob. The handle affords pulling [91] (figure 33).

![Figure 33: Affordability in real-world objects. The metal plate on the door to the left makes it obvious that the person has to push the plate to open the door. It affords pushing. The door to the right affords pulling.](image)

Similarly, affordance applies to many objects and materials around us. The term refers to the perceived and actual properties of an object, primarily those very fundamental properties that determine how the thing could possibly be used. Affordances hence give us strong indications on how to operate things. Plates are for pushing, knobs are for turning, balls are for throwing or bouncing and slots are for inserting things into [91].

The term affordance translates well to the human-computer interface world. It was originally invented by perceptual psychologist J. J. Gibson in 1977 to refer to the actionable properties between the world and an actor [92]. Gibson intended the term to span all “action possibilities” latent in the environment, objectively measurable and independent of the individual’s ability to recognize those possibilities. He also meant the action possibilities were dependent on the capabilities of the actor, such as the actor’s shear physical possibilities and constraints. In 1988 cognitive scientist Donald Norman introduced the term in Human-Computer Interaction and it is now known under the term perceived affordance in the software interface world. Affordance is as mentioned the perceived and actual properties of any object that determine how it can be used. The main difference between affordance and perceived affordance is subtle, but important, and the two terms do not necessarily refer
to the same thing [92]. Real-world objects emit both real and perceived affordance. A chair, for instance, affords sitting on, standing on (to e.g. change a lightbulb) or jamming a door open. It affords moving across the floor, but not if the chair is bolted down (which is then called false affordance [101]). In software user interfaces, the only thing the programmer or designer has control over is the perceived affordance. This includes the display screen, the keyboard, the pointing device and the selection buttons because they afford touching, pointing, looking and clicking on every pixel of the display. In effect, in reality most affordance in computer screen based interfaces is of no value [101]. For example a computer screen affords pushing with the fingers, but nothing will happen unless it is a touch screen. A graphical object on the screen will also afford clicking. The question is whether the user perceives this affordance; if he or she recognizes that clicking on the screen is a meaningful and useful action. Norman’s perceived affordances are thus a very subjective concept, but they are much more pertinent to practical design problems [101]. Both real and perceived affordances are described in detail in Donald Norman’s famous book The Design of Everyday Things [46] [91].

Norman claims that the way a user will understand how to operate a novel device is dependent on three concepts: conceptual models, constraints, and affordances [100]. Conceptual models were covered in section 5.2. Norman’s constraints are perhaps the closest real affordance gets to a user interface. They can be compared with restricting certain user interface actions depending on the context, such as limit the possible movement of the mouse. The convention that the mouse cursor is not able to move outside the physical borders of the screen is a constraint.

Perceived affordances in graphical user interfaces are subjective recognitions of affordance found in the real world. About fifteen years ago GUI elements got a new look. Using shades of grey, buttons and other controls suddenly seemed to pop out of the screen. This 3D look was not simply to look fancy, it added perceived affordance to controls. The raised buttons looked pushable, they afforded to be pushed [38]. Yet, designers often claim that when they have put an icon, cursor or other target on the screen, they have added affordance to the system. This is a misuse of the concept [100]. Affordance will exist independently of what is visible on the computer screen. The display itself is not affordance, it is the visual feedback that advertise the affordance: it is the perceived affordance.
Similarly, scrollbars and other elements in an interface emit perceived affordance. However, these affordances are to a large degree culturally constrained. As an example, the function of a scrollbar might indicate that it scrolls the window because it’s simply what the user is expecting. Whether it’s supposed to scroll up or down is a matter of consistent programming conventions and this is sometimes violated by programs that scroll in the “opposite” direction. Hence, the perceived affordability of scrollbars is a cultural, learned convention [92].

A good example of affordability is the “tabbed dialog” (figure 34).

![Figure 34: A tabbed dialog window from a Macintosh environment.](image)

In this window it’s really obvious that there are six tabs and that clicking on a tab header will raise the tab page to the top. When Microsoft usability tested this interface, usability measures went up from about 30% (using scrollbars) to almost 100%. Nearly every user they tested was able to figure out how to use the tab pages [38]. In addition to boasting great affordance, the tabbed dialog is also a very good metaphor.

**Consistency**

Before de facto standardized GUIs, every program had to reinvent the very fundamentals of user interfaces. This inconsistency was naturally an inconvenience for both users and programmers. Today, creating a usable application in many respects means emulating similar existing software as closely as possible. When an application looks and feels like
software the users have tried before, the usability threshold instantly becomes a lot lower. This really boils down to making the program model match the user model, as mentioned earlier. The user model will most likely reflect how the users perceive similar products and the users will expect the programs to behave in similar manners.

Some might find the above discouraging and limiting for creativity. In fact, some applications have seen great success despite differences from the common group of programs. Yet, there are at least two good reasons why it’s a good idea to imitate the look of similar applications, in general meaning Microsoft software:

- Even if it isn’t right, if Microsoft is doing it in a popular program like Excel, Word, PowerPoint, or Internet Explorer, millions of people are going to believe that it’s right, or at the very least fairly standard. At this point they will assume the new program works the same way.

- Microsoft spends ridiculously large amounts of money on usability testing and they keep detailed statistics on millions of tech support calls. On average 900 people are tested in the Redmond Usability Labs each month [48]. If Microsoft made a piece of software in a certain way, there’s a good chance they did it because more people can figure out how to use it that way.

The bottom line here is not to copy other programs in every single manner. On the contrary, new software can be like a fresh breeze and introduce interesting new concepts. The clue is to walk those new paths without breaking the rules of consistency. If for example a program allows for saving files, both CTRL+S, ALT-F-S, ALT+F-S (holding down the ALT-key while pressing F), and the floppy-button should work to keep things coherent. Many of our behavioral patterns when interacting with a computer have subconsciously become a part of the way we work. In many respects these patterns will function as a framework for usable software and should not be violated.

5.8 Microsoft Guidelines for Good Interface Design

Every time Microsoft announces a new operating system, they also create a set of guidelines for programmers to follow. These guidelines are based on extensive usability testing and
research and are meant to also create consistency across programs when third-party software is developed. They fairly much coincide with the reasoning above, but also add a few other interesting points. Below is a short summary of a few important guidelines from Microsoft [40].

Forgiveness and Feedback

A system should be based on forgiveness. This means that the users should be allowed to trial and error without risking to ruin things. An “undo” operation should be supplied and whenever hazardous operations are being performed, a warning should be issued. General feedback should be adjusted to fit the operation in question, depending on how critical the operation is.

Characteristics of Users

As already mentioned novice users have trouble understanding click-drag and double-click operations. These operations should hence be designed otherwise.

The same user group also generally has trouble understanding hidden windows and file management. Hidden windows are perceived as nonexistent and the file hierarchy is perceived as complicated. Also average users display trouble with file management. They understand the hierarchy, but will have trouble with e.g. moving and copying files.

Advanced users want efficiency. There is a challenge in creating an efficient user interface for advanced users without making it difficult for novice users. A solution might be to provide shortcuts.

Visual Design

The placement of “information objects” is the most important design principle. The user’s attention is first directed towards the top left corner, and then along the menu/toolbar. Gestalt principles [49] also show that our attention is drawn towards colored objects before black/white ones, isolated elements before grouped ones, and graphical objects before text. This placement paradigm hence implicates that objects with similar properties should be grouped together to provide overview.
3D effects must be used sparingly as they might indicate interactivity like affordances.

**Colors and Fonts**

It’s difficult to display connectivity and grouping using colors alone. Colors should thus be used as a supplement, not as the main information bearer. Also, only few colors should be used. Too many colors will make it difficult for the user to associate objects with the different colors. The colors should not be too sharp, and inverted colors should be avoided as they can make it difficult for the eyes to focus. In addition to this, colors are a matter of taste. What some users like others might dislike. Some colors can furthermore have special a cultural meaning, not to mention that about 8% of the male population suffer from different kinds of color blindness [51].

The interface should use clear and visible fonts. Fonts in italic will often be rendered blurry and should thus be avoided. The interface should be consistent and use as few fonts as possible. Using many fonts will seem very unprofessional. The usage of fonts that are already default in the system and similar applications are preferred.

**5.9 Customization and Tailoring**

Customization is to some degree common in many programs today. Anette Wilberg describes four rules for customizing software [5] [89].

- **Deletion.** The user can remove elements in the software. This can be elements or functionality that is annoying the user.

- **Superordination.** The user can combine several sub-functions into a superordinate function. This can for example be done by recording a macro in Word.

- **Selection.** This means the creation of new functionality by combining pre-existing functionality, yet without actually writing additional program code. An example can also here be recording a macro in Word.
• **Invention.** Invention is the process of creating/inventing new functionality based on existing functionality. Invention differs from selection because it’s necessary to add code.

Parallel to customization exists the related term *tailoring*. This is the process of adaptation of generic software applications to the particular work practices of a user organization [89]. Henderson and Kyng identify three types of tailoring or design-in-use [80]:

- Choosing between anticipated behaviors. An example of this is setting the default font of a word processor.
- Constructing new behaviors from existing pieces. Also here recording and using macros is an example.
- Modifying the artifact itself. An example here is reprogramming.

Common for both definitions is that they allow for the users to adapt their working environment to their needs.

The Switch project has not added many customization or tailoring possibilities. A few exceptions are the look of the slides list, the possibility to change default graphics folders and default settings for slide design. The usability testing phase of the development cycle does not intend to research customization and tailoring, but the test phase will however make note of if the users explore these features by their own initiative.

This does, however, provide a possible challenge for future work. Upcoming versions of Switch could indeed include possibilities for customizations and in this context a foundation for this work is already included with the property grid. One can imagine the property grid being used as a tool for customizing the attributes of the interface themselves, in the same familiar manner as the user edits all other attributes of presentation objects. This is in fact not a very far fetched idea and it is quite similar to the way Microsoft Visual Studio lets you manipulate the interface elements in the RAD designer. Naturally, quite some thought need to be put into how to layout the elements and what interface attributes should be available, and this is as mentioned a challenging task for the upcoming versions of the program.
6. The Design of the Switch User Interface.

In this chapter, the design and design choices of the Switch user interface will be presented. Many of the theories described up until now will be applied in practice. The chapter will focus on explaining that the position and design of every single element in the user interface has been carefully thought through, and that there is a reason for why every element looks and behaves the way it does.

6.1 The Major Elements of a User Interface

User interfaces commonly adhere to certain rules or conventions. These guidelines are often specific for the operating systems and there are for example different guidelines for designing applications for Windows, Macintosh or Linux. Yet, the distinctions go even deeper. As it turns out, practically every major release of an operating system comes with a unique set of guidelines and also most windowing themes within Linux come with their own guidelines for usability. Examples are Apple Human Interface Guidelines [56], KDE User Interface Guidelines [57] and GNOME Human Interface Guidelines 2.0 [58]. Even the Java programming language defines its own guidelines to make the look and feel of Java applications similar and distinct [59].

Luckily, there are many similarities between the guidelines. These similarities usually automatically provide a higher level of usability because software consistency, as described in chapter 5 [38], or cultural conventions [100]. The users will hence easier relate to the different parts of the user interface because the elements are recognized across several programs, operating systems or Linux window managers.

The Switch user interface client area is split into six different areas (figure 35). These are the menu, the toolbar, the slides list, the canvas (work area), the property grid, and the status bar. Mauro Marinilli identifies these same areas in his book about usability and layout [59]. However, he uses the terms “the upper area” (menu and toolbar), “the status bar”, and leaves the rest to the creativity of the designer. He also states that the selection area is the name of the area placed to the left, and the work area is the main area where the user’s attention is focused most of the time.
Figure 35: A screenshot of the Switch user interface.
6.2 The Menu

The menu of Switch is located topmost in the application (figure 36).

![Switch menu](image.png)

*Figure 36: The Switch menu (faded out for readability purposes).*

From the Windows User Interface Guidelines [16] a menu is to be displayed directly below the title bar. A standard Windows menu component is used in the Switch application, so it exhibits the same behavior as users are used to from other Windows applications.

To provide consistency, standard menu categories such as File, Edit etc have been used. This is coherent with the Windows guidelines and stressed for a clean user interface [60]. The menus also follow a convention of having the most important functions furthermost to the left, as this is where the user’s attention lingers first [40].

Nearly all menu choices provide a shortcut. This makes advanced users able to use the application quicker since hitting a shortcut is faster than using the mouse to perform simple actions [40] [65].

6.3 The Toolbar

The Switch toolbar is located directly below the menu bar (figure 37). The toolbar contains a set of the most frequently used functions the users need access to, and these functions are also mostly available in the menus either directly or through dialog boxes.

![Switch toolbar](image.png)

*Figure 37: The Switch toolbar.*

In the western world, people have a tendency to look to the top left corner for important information [40]. It is therefore natural to put buttons that represent prioritized tasks in the
leftmost part of the toolbar. In the case of the toolbar above, the button for New slide... is the leftmost button, probably one of the most common tasks. The toolbars are supposed to represent repetitive ongoing operations [60].

There is also a need for grouping toolbar elements. In the Switch toolbar, this is handled by putting related buttons next to each other. The Windows Vista guidelines define another two possible levels of grouping that can be used to emphasize relationship between controls [61]. By differencing between strongly and weakly related controls, subtle visual cues hint about how strongly connected different interface elements are. Weakly related controls can hence be represented by using space and possibly a separator line, and strongly related controls can use additional graphical effects called aggregators (figure 38).

![Figure 38: Aggregator in Windows Vista.](image)

As of now, aggregators have not been used in Switch, but are probably subject for future versions of the program.

There are two controls that are worth special mentioning when it comes to the Switch toolbar. First of all, a custom color dropdown menu has been developed to provide quick access to colors. Since slide appearance editing is a common task in a slideshow program, a new color dialog was created (figure 39).

![Figure 39: The Switch color dialog.](image)
There are several aspects to this control. First of all it contains access to a wide range of commonly used colors (in the center). Second, the left hand column represents the 16 standard original Windows and HTML colors [62] [63] (minus the grey shades). Third, the rightmost color column is a representation of common shades of grey. Fourth, the five most recent colors are represented in a most-recently-used queue in the bottom right corner. And fifth, should the user want other colors, he or she can click the “Custom Colors” button and a more complex color dialog box appears. From here the user can also define custom colors that are persisted to the bottom left row. Usability tests showed that the users easily handled the color dialog. As for consistency, the color buttons that open the color select dialog will probably appeal to most people that are familiar with Adobe Photoshop or Illustrator (figure 40).

These buttons hence help providing a unified and consistent work environment because of their resemblance to similar controls in other programs [40]. At the same time, they are also some of the largest controls on the toolbar. This is justified by Fitts’ Law since the color buttons are probably used a lot in slide design [42]. Finally, their unusual look is enhanced visually for novice users by providing a different cursor when the mouse hovers over the color buttons.

The second control in the toolbar that needs special mentioning is the font dropdown combo box (figure 41).
First of all, it displays the *look* of the font, not just the name. This gives the users a first cue of what the font will look like in the slideshow. Many other programs simply display the font name and leave it to the user to remember what the font looks like. Second, the dropdown area is a lot *longer* than in many other common programs. Joe Spolsky argues that most programs don’t take advantage of the available space for dropdown controls [38]. The result in these programs is that if the user has to scroll to find a font among many installed fonts, he or she has to click on the tiny scrollbar in the font dropdown many times to find the wanted font. If the user decides to drag the scrollbar, it probably gets even more difficult since the fonts will race up or down at a high speed. On top of this, as already discussed in section 5.5, many users find it difficult to control the mouse.

It should be noted that the functionality of the toolbar has slightly been altered in the Switch user interface. Whereas the toolbar in e.g. Microsoft Office lets the user select something and then change it using the toolbar, the Switch toolbar is intended as the settings for *new* slideshow elements and the property grid is the place to change settings of elements. The usability tests will give an indication of whether this is appropriate.

It should finally be mentioned that all controls on the toolbar provide tool tips, i.e. a small informational text if the mouse pointer is held over a control for about a second. An example of a tool tip is depicted in figure 2.
6.4 The Slides List

On the left hand side of the application is the slides list (figure 42). Just like in other presentation programs, the slides list represents all the slides in the slideshow.

Figure 42: The Switch slides list (shortened for appearance purposes).

A corollary in user interface design states that there is no need to demand that the user should remember something that the computer can remember for them. In fact, people also remember a lot better when they are given a few clues and that they would always prefer to choose something from a list than recall it from memory [38]. The slides list is hence critical for the navigation between slides since it displays small images that represent the slides in the slideshow [39].

As an attempt to improve usability, the Switch slides list has two special features. Just like in the Macintosh Dock (figure 16), the list enlarges the currently selected slide. The other slides are gradually minimized in size the further away from the selected slide they are. A key idea behind this implementation is that the user can have more slides visible in the slides list at any given time. Most probably, the user has some general clue of what the contents of the individual slides are, and hence clicking it will make it bigger and thus more readable.

A second feature in the slides list is that it enlarges the slide the mouse hovers over. That way, to inspect the slides, the user simply has to move the mouse up or down along the list to find the wanted slide. Naturally, these two features can be switched off and leave the list in a static state. The default setting should be off.
Customization of the slides list was not subject of the usability testing. Yet, the general feedback from the users was that it enjoyed the feature and found it practical.

6.5 The Canvas

The canvas is the main working area in the application. This is where elements are added, removed, drawn, and selected for editing. By using the mouse, the users can also move and resize elements. All new slides and slideshows are created based on a task-oriented approach, and are hence created with initial content such as a heading and a content textbox.

As already mentioned, one of the critical choices in the development process was to represent selecting of the graphical elements somehow. A selected element in a Switch slide is depicted in figure 43.

![Switch 1.0](image)

*Figure 43: A selected textbox in the Switch user interface marked with a hatched bounding rectangle and eight grab handles.*

The mouse cursor will furthermore give visual cues of which operations are allowed. When the cursor for example hovers over a grab handle, the cursor will look like a resizing arrow, and hence give the user some visual feedback of the possible operation.

Whenever a user clicks any object on the screen, its properties will be displayed in the property grid to the right. This is a fundamental concept in the Switch user interface.

6.6 The Property Grid

Usage of the property grid in Switch was to a large degree motivated by two factors. First of all, the program 3D Studio Max has used a similar panel with tools for years [64]. However,
this panel was not a grid like the property grid, but more like a panel of customized tools and controls specialized for each graphical object (figure 44). The second factor of influence was Microsoft Visual Studio. Visual Studio’s RAD designer has used the very same property grid for many years and this control has proven very useful in application development.

The property grid (figure 44) has already been described in section 2.1. It provides a set of attribute-value pairs that the user can inspect and edit. Whenever a value is edited in the property grid, the canvas element is immediately updated, and a main theory was that the usage of this grid would reduce the need for dialog boxes. Dialog boxes have the unfortunate property that the user cannot usually see the settings he or she changes until after the user has clicked the “Ok” button. If they were wrong in their guesses, the dialog box has to be opened again and the procedure repeated until the graphical object looks the way the user wanted. By omitting these steps of opening and closing dialogs, a hypothesis was that the users would be able to work faster.

![Property Grid](image)

**Figure 44: The Property grid in Switch, and the 3D Studio MAX counterpart.**

As described earlier, a property grid has the ability to display custom UITypeEditors. These editors provide a visual means to input values without cluttering the interface with dialog boxes. Switch uses these editors extensively. More images and examples of the property grid and the UITypeEditors can be found in chapter 2.1.
A handy feature about the property grid is that it provides a lightweight help/info function. At the bottom of the control a text field is provided that can display a short and informative text. This text field will change to display a property description whenever the user clicks on a property. Unfortunately, many users don’t notice the text or simply don’t bother to read it [38]. On the upside, the feature provides a handy information function that doesn’t disrupt the user in the work like a dialog box or a message box [39].

As a downside, providing descriptive attribute names is not done easily. The attribute names reflect the C# variable name in the programming code and can thus not contain spaces or other special characters. For that reason, that step has been omitted at this point. Future releases of the project will for sure provide more readable attribute names.

6.7 The Status Bar

The final main element of the interface is the status bar. It is located at the bottom of the application and is used to display short, non-critical information to the user. The status bar should never be used to display critical information to the user since it might not be likely that the user will notice information displayed there. This is fully coherent with the Windows Guidelines about hierarchy of information [40]. Information that will be displayed can be such as the mouse cursor coordinates or a progress bar indicating that the program is working on something in a separate thread (like generating the slides in flash). Halting the rest of the program as a result of doing a processor intensive operation can lead to the user believing the system has hung. Complicated computations should be run in separate threads and information is given through controls in the process. This gives the user a favorable perception of usability and performance [66].

6.8 Settings

As already mentioned the program should remember values and thus relieve the user from keeping track of settings [39]. Switch will go to the extent of keeping track of the user’s last files used, window location, window size, custom colors, slides list settings, as well as default values for slide layout, and so on. This information will then be used at the next start-up of the application to position and size the window.
7. Software Usability Testing

The purpose of usability testing boils down to the end user experience. It all comes down to how the users interact with the software and how they feel about it, for a product to be usable. But what is really usable? How can we quantify usability goals like easy to use, easy to learn and fun to use?

7.1 Defining Product Usability

Program development today is not the same as ten years ago. The increase of processor speeds, computer memory, network speeds and so on has given software developers a wide range of new tools. This has led to the fact that most products that are developed today are networked, mobile, database-linked, web-linked and sophisticated products for users of many skill levels [26]. Software developers have to walk the line between business processes, new technology, intuitive interfaces and performance to give the users what they want, when they want it and how they want it. Usability testing is the only tool that can provide a path to this consistency. At this point we can consider usability testing the glue that keeps these elements together (figure 45).

Figure 45: How aspects of usability fit together.
Usability is quite often referred to as a *subjective* notion. The term *user friendly* is on the other hand strongly coupled with usability, but neither do really have a meaning unless they can be quantified, measured and tested. Usability testing is based on the process of defining which measures are needed to define product usability. These measures are then used to evaluate how well the users interact with the product and how well they perform their tasks.

ISO\(^1\) has two definitions for usability:

1. A set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users. [24]

2. The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use. [25]

These definitions underline the argumentations that have been presented so far.

To understand what a usable product is, testing is necessary. This is where usability testing comes into the picture. However, it’s important to realize that usability testing comes hand in hand with good design. Both are needed to create a good product. To quote Theo Mandel: *Good interface design does not guarantee a usable product, while user interface testing can never substitute for good design. They are both part of the interface design process called usability engineering* [26].

So basically, if the goal is to want to make a user friendly product, it’s necessary to have both a good design and the right usability testing procedures. The problem is, however, that the term *user friendly* is usually referred to as so lacking in content and specificity that it is commonly quite meaningless. Anderson and Shapiro go on and propose several categories that can be used to get a better definition of the term [27]:

- Easy to use
- Easy to learn (and teach)
- Easy to relearn
- Easy to unlearn

---

1 The International Standards Organization
• Easy to avoid harm
• Easy to support
• Easy to audit
• Easy to share within a group
• Easy to integrate into existing operations

These categories can be thought of as helpers for any usability testing definition phase. They can also be used to create questionnaires, checklists, or guidelines for evaluation of products. The ongoing process in this chapter will among other things use many of these categories as a foundation.

7.2 Defining Software Usability Testing

At the most basic level, usability testing will uncover whether the users can use a product. There are many reasons for testing; Theo Mandel lists the following [26]:

• Designer’s and developer’s intuitions about a product aren’t always correct.
• Designer’s and developer’s terminology doesn’t always match the user’s.
• People differ and therefore there is no “average” user.
• Usability design principles and guidelines aren’t sufficient.
• Informal feedback is inadequate for product evaluations.
• Time, money and resources spent on usability evaluations are worthwhile.
• Products built in pieces will usually have system-level inconsistencies.
• Problems found late in the development process are more difficult and more expensive to fix.
• Problems fixed during development will mean reduced support costs later.
• Usability evaluations can provide advantages over competitive products.

Usability testing is the process of evaluating product designs (paper designs, prototypes and final products). The purpose is to provide feedback to improve product design, to reduce bugs and problems, to compare products and versions, and to validate that products meet the predefined usability goals and objectives. Usability testing can thus be everything from observations, interviews, surveys, qualitative and quantitative analyses, contextual inquiries, heuristic evaluations, focus groups to advanced laboratory testing. Test participants should also be good representations for the actual target user groups to avoid
bias, and the tests should take place in an environment as close to the realistic environment as possible. Also refer to chapter 4 about scenarios and program goals and objectives.

Usability testing commonly focuses on performance measures and subjective measures. Whether to weight one more than the other is a question answered based on the product goal and objectives. The two measures are defined the following way [26]:

- **Performance measures**: The counts of actions, task completion, task time, errors, and frequency of assists. How much the user remembers of the usage of the interface after the test can also be included here. These measures are also called quantitative measures.

- **Subjective measures**: Perceptions, opinions, oral and written data collected from the users, judgments, preferences, and satisfaction level. The data is collected based on the users’ interactions with the system and their own performance. These measures are also called qualitative measures.

### 7.3 Usability Test Participants

As already mentioned, the usability test participants should be users that are as close as possible to the target user groups. For the sake of this project, much of this work was already done in chapter 4, where *novice users* and *advanced users* were defined.

The next natural step is to find out how many users are needed for each test. Jakob Nielsen, a former researcher at Sun Microsystems, popularized in the early 1990s the idea that only a few test subjects (typically five) are enough. There was simply no need to push a whole range of users through the testing of an interface that had its obvious flaws. The mathematical model of “five users are enough” was later described by mathematician R. A. Virzi, and Nielsen’s work became known as “heuristic evaluation” [28].

Nielsen’s and Virzi’s work was based on the statistical binomial formula:

\[ U = N (1 - (1 - p)^n) \]

In this formula, \( N \) is the total number of usability problems in the design, and \( p \) is the proportion of usability problems discovered while testing a single user. The formula yields
the number of usability problems found in a usability test with $n$ users. According to Nielsen, from a wide range of experiments, a typical value of $p$ is 31% [29]. The corresponding graph is depicted in figure 46.

From this model it’s easy to see that zero test users provide zero insight. As soon as a single test user is tested, almost a third of the problems are found, and with five users, almost 85% of the problems are discovered. This is because there is some overlap between what you learn in each case. The model also shows that 15 users are enough to uncover almost all the problems a user interface might have. Nielsen goes on to describe that it’s a far better idea to have three iterations where five people are tested in each round, and refine the user interface between each iteration, than to test 15 people in one iteration (unless time and resources are not an issue). Also Theo Mandel claims that four to eight test participants are usually enough to uncover most major usability problems [26].

In this project, two fairly distinct user groups have been defined. Even though they are different, there will probably still be some overlap between what it is possible to learn from the test participants. After all, the users are still human. For this reason, Nielsen recommends that 3-5 users are enough in each iteration.
Nielsen’s model has been eagerly questioned by more advanced mathematical models and empirical studies. Still, three to five users will be considered adequate with each iteration of testing with each user group in this project.

7.4 Product Usability Goals and Objectives

It’s not uncommon that project leaders and developers define product goals similar to “our product should be intuitive and easy to learn”. Whereas such an objective may sound fine, it is not clear enough to serve as a product goal.

Usability goals and objectives should be defined for all software products. These are high-level statements of a product’s desired benefits in the areas of certain usability factors. Paul Booth defines four factors that comprise usability: Usefulness, effectiveness, learnability and attitude [30]. A description of the four usability factors is listed in table 7.

<table>
<thead>
<tr>
<th>Usability Factor</th>
<th>Description and Measures</th>
</tr>
</thead>
</table>
| Usefulness       | • The degree to which a product enables users to achieve their goals. (Can users use the product?)  
|                  | • An assessment of users’ motivations for using the product.  
|                  | • Usefulness measures are usually captured by performance data. |
| Effectiveness    | • How successful a product is in allowing users to perform their work. (How well can users do a task with the product?)  
|                  | • Effectiveness measures are typically captured through performance data. |
| Learnability     | • Users can use a product to a defined level of competence after a predetermined amount and period of training. (How well trained are the users?)  
|                  | • Learnability measures are typically captured through performance data. |
| Attitude         | • Users’ perceptions, feelings, and opinions about learning and using a product. (What are users’ thoughts about the usability of a product?)  
|                  | • Attitude measures are typically captured through satisfaction data (written and oral feedback). |

*Table 7: Factors that make up usability [26]*
The agile reader will recognize these four factors as the same factors that were used in table 3 in section 4.2 during the description of product goals and objectives in the four phase development process. It’s a natural step to return to these factors during the usability testing phase. As a side note, other usability authors have also categorized goals and objectives in a similar manner. Shackel proposes the following four factors for usability: learnability, effectiveness, flexibility, and user attitude [31].

It’s worth to mention that the usability goals by themselves are not directly measurable. These goals must further be broken down to more specific and detailed refinements. The most important key here is that the detailed objectives describe specific performance or actions, and that they are measurable given certain conditions.

### 7.5 Creating the Final Usability Tests

At this point, the work done in chapter 4 becomes clearly valuable. As already mentioned, the user scenarios and tasks created in section 4.2 would serve as the base for the usability testing in this chapter. The usability testing phase is where the work done so far comes to a full circle and the scenarios and tasks become part of the testing of the interface.

Table 4 in section 4.2 lists eight user scenarios with tasks, four for each user type. In the usability testing phase, these scenarios and tasks will be in focus as the users are being evaluated while using the software.

There are many ways to conduct usability tests. In the early stages of prototyping, more informal methods are not uncommon. This development process has been no exception and the early models have been tested through evaluations by fellow students, friends, family, co-workers and user-groups on the Internet. Many refinements and changes have taken place along the road. The different parts of the user interface have slowly come to life during a long programming and prototyping cycle, and at this stage it’s time to test how the system works together as a whole. This is the four-phase development cycle in practice.

More advanced usability tests are often conducted in usability labs. Test subjects are asked to perform a set of tasks using the software, and at the same time think out loud so that the usability test team can better understand the actions and thoughts of the test participants. In some cases, audio and video are recorded as well for further analysis.
For the sake of this project, the early tests will not be covered in great detail. Yet, several authors claim they are invaluable. Joel Spolsky, an author of a whole range of software articles and books describes the Hallway Usability Test in his article The Joel Test: 12 Steps to Better Code. The Hallway Usability Test is where the programmer grabs the next person that passes by in the hallway and forces him or her to use the code that was just written. Whereas this is very informal, executing these small tests on a few people and focusing on certain parts of the user interface will uncover a great deal of all the problems the user interface might have (Spolsky himself claims 95% of the problems will be found, but doesn’t back that number up pointing to scientific data). These tests are still a good source for making refinements [32].

Also Theo Mandel mentions early-phase testing under the concept heuristic evaluation [26]. The methods may vary and the process is often conducted by user interface designers and experts, but he stresses that they are not a replacement for laboratory testing. Nevertheless they often provide usable feedback in the development process.

As mentioned above, usability test objectives provide a measurable means to evaluate testing. The usability test objectives for the two types of users are described in table 8 and table 9.
# Usability Objectives Set One: Novice Users

<table>
<thead>
<tr>
<th>Usability Objective</th>
<th>Criteria</th>
<th>Performance</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Usefulness</strong></td>
<td>• 80% of the users&lt;br&gt;• Complete the test</td>
<td>• Successfully complete the test</td>
<td>• After 4 task scenarios</td>
</tr>
<tr>
<td>After 4 task scenarios, 80% of the users will successfully complete the test.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| **Effectiveness**   | • 80% of the users<br>• Complete each step of the test within 3 minutes | • Successfully complete the test | • After 4 task scenarios |
| After 4 task scenarios, 80% of the users will be able to complete each step of the test within 3 minutes. |

| **Learnability**    | • All users<br>• At least 60% of tasks similar to the ones in the scenarios | • Successfully attain product knowledge | • After 4 task scenarios |
| After 4 task scenarios, all users will be able to complete at least 60% of the tasks given to them in the test (and succeeding tasks). |

| **Attitude**        | • 85% of the users<br>• Satisfaction level of 7 or better on a 10 point scale | • Satisfaction rating | • After 4 task scenarios |
| After 4 task scenarios, 85% of the users will rate their satisfaction with the product at a level 7 or better on a 10 point scale. |

*Table 8: Usability objectives for each usability factor in the testing of novice users.*
### Usability Objectives Set Two: Advanced Users

<table>
<thead>
<tr>
<th>Usability Objective</th>
<th>Criteria</th>
</tr>
</thead>
</table>
| **Usefulness**      | • 100% of the users  
After 7 task scenarios, 100% of the users will successfully complete the test.  
• Complete the test |
| **Effectiveness**   | • 80% of the users  
After 7 task scenarios, 80% of the users will be able to complete each step of the test within 3 minutes  
• Complete each step of the test within 3 minutes |
| **Learnability**    | • All users  
After 7 task scenarios, all users will be able to complete 100% of the tasks given to them in the test (and succeeding tasks)  
• 100% of tasks similar to the ones in the scenarios |
| **Attitude**        | • 85% of the users  
After 4 task scenarios, 85% of the users will rate their satisfaction with the product at a level 7 or better on a 10 point scale  
• Satisfaction level of 7 or better on a 10 point scale |

<table>
<thead>
<tr>
<th>Performance</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Successfully complete the test</td>
<td>• After 7 task scenarios</td>
</tr>
<tr>
<td>• Successfully complete the test</td>
<td>• After 7 task scenarios</td>
</tr>
<tr>
<td>• Successfully attain product knowledge</td>
<td>• After 7 task scenarios</td>
</tr>
<tr>
<td>• Satisfaction rating</td>
<td>• After 7 task scenarios</td>
</tr>
</tbody>
</table>

*Table 9: Usability objectives for each usability factor in the testing of advanced users.*
7.6 The Conduction of the Usability Tests

The user tests in this project were conducted in an improvised usability lab at the Institute of Informatics at the University of Oslo. Using a digital video camera and a dedicated computer with Windows XP SP2 and a big screen, the users were tested and the sessions recorded. A digital timer provided means for measuring how efficient the users were in completing the tasks. The digital camera was aimed at the screen to record how the users performed on the tests and throughout the training (figure 47).

![Figure 47: The usability test setup. A digital camera behind the user records the screen.](image)

First, users of both groups were interviewed with a few simple warm-up questions to find out their skills and knowledge about computers and software. These questions are presented in
Appendix A.

Then the users were given a brief introduction to Switch, what it is and how it works. Since the users were of very varying skill levels, the time needed for this part turned out to be very different from user to user. They were also informed that the application is a prototype and that not all menus and buttons were implemented.

The third part of the test focused on having the test participants acting out the four scenarios described in table 4. While they performed their tasks, they were instructed to think out loud and describe each step they did and why they did it. All this information was recorded in the lab. At this point the users were allowed to ask as many questions as they wanted.

The advanced users were also instructed to act out some of the scenarios of the novice users since they were assumed to attain their basic computer and software knowledge, and more. This proved to be a waste of time since the advanced users picked up on the interface extremely quickly.

In the fourth part of the evaluation, the test participants were given a small “test” to see if the intuitive interface was easy to remember. The test was a combination of the skills elaborated on in the scenarios and is described in table 10. At this point the users were not allowed to ask any questions and were instructed to try things out for themselves.

Finally, the users were interviewed to get some input about their satisfaction with the product and to attain their attitude towards it.
<table>
<thead>
<tr>
<th>Novice Users</th>
<th>Advanced Users</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scenario: Test</strong></td>
<td><strong>Scenario: Test</strong></td>
</tr>
<tr>
<td>The user wants to create a fully functional slideshow with a custom title slide and with text and graphics</td>
<td>The user wants to create a fully functional slideshow with a custom title slide and with text and graphics</td>
</tr>
<tr>
<td>User tasks (in any order):</td>
<td>User tasks (in any order):</td>
</tr>
<tr>
<td>• Opens the program</td>
<td>• Opens the program</td>
</tr>
<tr>
<td>• Chooses a slide background color or image</td>
<td>• Chooses a slide background color or image</td>
</tr>
<tr>
<td>• Turns off background inheritance</td>
<td>• Turns off background inheritance</td>
</tr>
<tr>
<td>• Inserts a textbox for the title slide</td>
<td>• Inserts a textbox for the title slide</td>
</tr>
<tr>
<td>• Inserts the slideshow title in the textbox</td>
<td>• Inserts the slideshow title in the textbox</td>
</tr>
<tr>
<td>• Edits the textbox font properties</td>
<td>• Edits the textbox font properties</td>
</tr>
<tr>
<td>• Changes the textbox background color, border color and border width</td>
<td>• Changes the textbox background color, border color and border width</td>
</tr>
<tr>
<td>• Inserts slide 2</td>
<td>• Inserts slide 2</td>
</tr>
<tr>
<td>• Chooses a slideshow background color or image</td>
<td>• Chooses a slideshow background color or image</td>
</tr>
<tr>
<td>• Inserts a textbox in slide 2, edits the text and changes the font properties</td>
<td>• Uses the drawing tools to draw a star in slide 2</td>
</tr>
<tr>
<td>• Repeats for as many slides wanted</td>
<td>• Saves the slideshow</td>
</tr>
<tr>
<td>• Deletes slide 2</td>
<td>• Exports the SWF file</td>
</tr>
<tr>
<td>• Saves the slideshow</td>
<td>• Navigates to the directory of the SWF file using Windows Explorer or a folder browser.</td>
</tr>
<tr>
<td>• Exports the SWF file</td>
<td>• Opens the SWF file in Flash Standalone Player or Microsoft Internet Explorer</td>
</tr>
<tr>
<td>• Navigates to the directory of the SWF file using Windows Explorer or a folder browser.</td>
<td></td>
</tr>
<tr>
<td>• Opens the SWF file in Flash Standalone Player or Microsoft Internet Explorer</td>
<td></td>
</tr>
</tbody>
</table>

Table 10: The test scenario for novice and advanced users.

As mentioned, the users’ skill levels varied greatly. Five users were tested and they ranged from “fairly new to computers” (2) to intermediate (1) to experts (2). Naturally this had a huge impact on performance. While the two expert users only needed fifteen to twenty minutes for the whole testing and training phase, others needed up to one and a half hours. Furthermore, the novice users had also practically never seen a presentation program before. This proved a challenge. The intermediate candidate was placed in the “novice users” group, partly by own choice.
7.7 Test Analysis and Results

The test participants were as mentioned asked to perform a number of successive tasks. Transcripts of the usability tests are provided in Appendix B. Using a timer, the users were timed to record how long time they used for each task. This was a fair measurement of how difficult the individual tasks were. As mentioned in chapter 2, the main focus of the test was to see if the users understood the slides list, the main work area and the property grid. The toolbar was also given some focus although it was not the main test target. Interestingly, the toolbar proved to be one of the elements that confused the users the most.

The Slides List

All users were asked to perform operations on the slides list. These were operations like inserting and deleting slides, rearranging the slides, and giving focus to a slide to alter its properties in the property grid. These operations are available in a context menu that appears when the user right-clicks a slide in the slides list, through shortcuts, from the toolbar, and in the menu.

The two novice users turned out to have problems understanding clicking on the slides list with the right mouse button. Even though it was explained and described numerous times, they for some reason had troubles understanding that the two buttons do not expose the same behavior. This is completely in terms with Microsoft guidelines’ descriptions of users [40]. To be consistent with other programs, the operations on the slides list was also available in the menu, as a shortcut and on the toolbar. It turned out to be only the advanced users that tried them all, the intermediate (novice) user preferred the menu, and the two novice users fiddled with the context menu, most probably because the context menu was easier to remember from the training phase. None of them used the button for inserting new slide, which is interesting since it has been given the most prominent location on the toolbar. As a side note, PowerPoint has a big button with the text “New Slide” in the middle of the toolbar.

The Working Area

As above, the expert users had no problems handling the main working area. All objects provide grab handles for resizing, and a hatched outline for moving. It is also possible to
move the object by dragging it directly with the mouse. The expert users recognized these elements immediately.

The novice users had problems understanding the difference between a selected and an unselected element. They would for example try to move the object with a grab handle, only to be puzzled when it grew instead. These users seemed to believe the grab handles were some sort of decoration. Another action that proved difficult for the novice users was to insert new items. This was typically done by selecting a textbox from the toolbar (pushing a pushbutton in) and then clicking and dragging with the mouse on the canvas. The novice users would not understand the difference between a button (in reality a checkbox) that was pushed in and a button that was in the “out” state, often resulting in doing something on the canvas and nothing happened. They would also typically release the mouse button after only a few millimeters, resulting in micro-textboxes that were hardly visible, only to believe that this clutter on the canvas was some sort of program error. Another attempt they made was to drag the button itself onto the canvas, resulting in confusion nothing happened and the button again went from on to off. Clearly the user interface can be improved here by providing some sort of foolproof method of not creating such micro-boxes. It could also be improved by using stronger coloring to demonstrate that a button is pushed in, and a means to detect if a user drags from the toolbar onto the canvas.

Another mildly confusing item was the textbox itself. To edit the text in a textbox, the users had to double-click inside the textbox. The novice users did not understand the concept of double clicking very well, another issue that is explained in the Microsoft Guidelines [40]. PowerPoint uses here only single clicking, which is perhaps the best way to go. Furthermore, when the users were asked to change font colors and sizes, they tended to select the text using the mouse and then go directly to the toolbar and edit the values there. This is the common way to do it in Office programs. As explained earlier, the Switch toolbar has a somewhat different purpose than the Office toolbar. Its main function is to provide the settings for new elements, where the property grid is the place to go to edit the attributes of existing elements. Perhaps a better solution here is to provide some sort of hybrid solution where selected text is a special case. Also the expert users did this, although only once each.
An interesting thing to note was that neither of the users attempted the “de facto” way of editing an object. De facto means in this context to right-click an object and open a dialog box where all of that object’s values are presented. Hopefully this is an indication that the property grid provides the means to edit those values without the need to go through a dialog box.

As a trivial afterthought, none of the users questioned the absence of bulleted lists. This is probably to the satisfaction of Edward Tufte, who has been fighting the lack of rhetoric content in such lists for years [81].

The Property Grid

The property grid turned out to be quite understandable, perhaps on the contrary to prior beliefs. As already mentioned, the novice users had some trouble understanding the concept of “selected item”. One of the key ingredients in the Switch user interface is that whenever the user clicks on something, its attributes and values will be displayed orderly in the property grid. However, if the novice users had just changed the font settings of a textbox and were asked to change the slideshow background color, they were puzzled why the slideshow properties were not automatically there. On the other hand it seemed to provide an easy way for the users to get acquainted with the properties of the different elements as all attributes of an item was categorized and presented in one place. The novice users frequently scanned the properties up and down in order to solve the tasks. They also seemed to believe the solution to the given task always lay in the property grid, which sometimes led to confusion when the solution was elsewhere. The advanced users had seen property grids before and had no problems relating to it and use it.

One of the goals of the testing was to see if the users would understand the different custom UITypeEditors described earlier. These editors are provided for most values to simplify the selection process and reduce the possible numbers of errors. As it turned out, the editors provided were quite easy to understand. The only possible exception was that to enable an editor (i.e. make it pop up), the users had to click on a small button inside the property grid, which only appeared after that particular attribute or value had been selected in the grid. This is the standard behavior of the control. An improvement here could be to make the editor appear whenever the user clicks anywhere inside the value field.
In fact, the extensive usage of type editors made it difficult for the novice user to understand when it was not there. For example, when they were asked to change the value for “border width”, an integer value, the property grid was set to accept a written input value in the text part of the grid, but the users more than once expected a pop-up dialog to appear. A solution to this might be to provide both a type editor and a means to input text values directly into the grid. In fact, the color properties that supported the Switch custom color selector allowed the users to either click on a color button in the selector, write the textual name of the color directly in the grid (i.e. “red”), or write the hex values of the colors in the grid text field.

Another interesting comparison was the evaluation of the color selector provided as standard with the property grid (figure 48) and the color selector developed for the Switch project (figure 39).

It turned out the users liked the way the standard color selector presents the colors and the names, but that they had problems finding the colors they were looking for. Few of the users also noticed that there were actually tab-sheets on the top of the dialog. In comparison, the Switch color dialog (figure 39) provides many colors without tab-sheets and scrollbars, but neither of the users tried to explore colors beyond the ones presented in the control.
Other Controls and Objects

One major problem evident in all tests was the ability to understand the difference between slide colors/background image and slideshow colors/background image. The appearance of the slide is governed by a variable that turns inheritance from the slideshow settings on and off. Nearly all users felt this was awkward, especially since they could set a slide background color but nothing would happen (since the slide was set to inherit from the slideshow by default). A possibility might be to skip the background settings in the slideshow altogether and rather have a function that can set the colors and backgrounds of the slides in a bulk.

Limitations of the Study

Since the project is still under development, some of the features in the interface have not yet been implemented. This might have lead to confusion although the users didn’t seem to explore these features on their own.

An obvious downside to the property grid is the fact that it cannot easily display “readable” attribute-value names. It is designed to display variable names, and these names are for instance not allowed to contain whitespace. Since the space occupied by the property grid is by default not overly wide, variable names such as “SlideShowBackgroundColor” partly becomes covered by the frame and is difficult to read. The novice users tended to scan the property grid a lot, and changing to more readable names would definitely have an advantage. This is scheduled for future versions, however.

Another limitation might have been that the program interface is in English. English was not the users’ first language and this increases the possibility for misunderstandings.

The novice users spent a long time learning the basic keystrokes and mouse gestures of the program. This might have led to the users becoming impatient and thus not focused. Ninety minutes is a long time to sit in front of a computer if it is not someplace you are used to sit.

Four of the five test subjects were friends or family. This might have led to them answering questions more in favor of the program than they normally would.
The users have different experiences with computers. Ranging from practically nil to being experts, it’s difficult to put the users in only two categories.

During the conduction of the tests, it might have been favorable for the candidates that the assignments were presented orally. Even though it was not intentional, subconscious tonality, approving/disapproving looks in the eyes etc could have led to the assignments being solved easier.

It was probably unnatural for the users to think out loud while they tried to operate the program. This can have made them uneasy.

The users were not in their own work environments. The computer was different and the operating system might have been different.

**Usability Test Objectives Results**

In section 7.5, various test objectives were presented. As the tests were conducted, these criteria were continuously measured since they present a more measurable way to evaluate the application. The following paragraphs will present the results of these measurements for the novice and advanced users respectively.

**Novice users:**

**Usefulness:** 100% of the users completed the test successfully.

**Effectiveness:** 100% of the users completed each step of the test in less than 3 minutes, although the two most novice users used 1-3 minutes on each task.

**Learnability:** The users completed close to 100% of the tasks given to them. The rest was completed with a minor hint.

**Attitude:** Two of the users rated the program to 6.3 and 7 respectively on a scale from 0 to 10. The final novice user felt it was too difficult to rate the program after only spending a little time with it and not knowing about the alternatives.
**Advanced users:**

**Usefulness:** 100% of the users completed the test successfully.

**Effectiveness:** 100% of the users completed each step of the test in less than 3 minutes, typically in seconds.

**Learnability:** All users were able to complete 100% of the tasks given to them.

**Attitude:** The users rated the program with a satisfaction level of 7 and 9 on a scale from 0 to 10 respectively.

From the above it’s possible to conclude that the tests yielded success given the criteria, objectives and scenarios. However, as a final thought it might be reasonable to question whether the tests or scenarios were too easy. The 3 minute time limit on most tasks was offered as an example by Theo Mandel [26]. Nonetheless, 3 minutes seems like a very long time for a user to explore the user interface in order to complete the different steps.

**Conclusion**

The users seemed to find it easy to use the interface after some training. Even though the users had widely different backgrounds, they were all eventually able to navigate the user interface and complete the tasks. The property grid was perceived as providing a good overview and was in itself quite easy to understand. Several of the users said it was easier to use than PowerPoint.

The function of the toolbar was somewhat confusing. Since the users had previous experience with Microsoft Office, they tended to believe the Switch toolbar, which looks somewhat similar, behaves the same. Because of this, the toolbar’s main function was not overly intuitive.

The slides list was eventually readily understood by all users. Manipulating the slides list was enabled through both buttons, menus, shortcuts and context menus, and all users found a way to work it, although no one way proved more evident than others. The “New Slide” button was mostly overlooked.
As for the main work area, it proved somewhat challenging for the novice users. They had problems understanding how to resize and move objects, and to create new objects. The textbox in particular was perceived to work as a text editor. The novice users also had problems understanding when to right-click, left-click and double-click items, and what the difference between these mouse gestures are.

All users found the concept of slide background vs. slideshow background, and background inheritance confusing.

**Test Reliability and Validity**

Test *reliability* means in short that a test can be repeated a number of times and will still yield the same results [26]. The testing phase given the objectives, conditions and proper training is in this case believed to produce similar results again. Yet, this can only be proven by testing anew.

Test *validity* means that the test accurately measures what it was supposed to measure. After the experience of actually conducting the tests, the general afterthought is that test criteria should have been stricter. Even though the interface is fairly complex for inexperienced users, three minutes is still perceived as a long time to scan the interface for hints on how to solve the tasks. There is still plenty of room for improvements on the user interface and the test criteria should perhaps harden along with the increase in usability.

**7.8 Other Sources of Usability Information**

At this point mainly articles, books, feedback and usability testing have been the primary sources of usability information. Since quantitative analysis methods as of now have not been used, this part of the specter is based on research and articles performed by others and referenced throughout the thesis.

**External Evaluation**

A completely different form of gathering usability information is to have the application analyzed by external professionals. In this context, human factors professionals and
cognitive psychologists can be crucially important for an interface design. For example, in 1991, Jeffries et al. performed a study where a software product was given to four different groups of testers [33]:

- Group one was four human factors specialists and was given two weeks to review the product.
- Group two was one user interface tester and six test subjects, where the six test subjects were observed as they used the software and the tester wrote a report on his observations.
- Group three was three programmers that formed a set of principles or guidelines that were then applied to the user interface.
- Group four was three other programmers that conducted a “cognitive walkthrough” on the product.

The software the different groups were analyzing was purposefully considered having several flaws. These core problems were already defined by a panel of seven interface design experts that also had evaluated the software.

When all tests were completed, the final test results were quite interesting. When they counted the number of problems found by each group, it turned out that the human factors professionals found almost four times as many problems (152) as each of the other groups (about 39 each). They also found three times as many core problems as the other groups (105 to 35). This is a strong indication to not count solely on usability tests. Human factors professionals and cognitive psychologists do indeed bring valuable information to the table.

**External Evaluation of the Switch User Interface**

As the Switch user interface started to become ready for testing, it was handed over to psychologist and HCI professional Ole André Solbakken. Solbakken is a former researcher at SINTEF in the Human-Computer Interaction Research Group, and has evaluated many interface projects such as video telephony systems and avatars. Based on former research and experience, Solbakken evaluated the software, wrote a report, and was then interviewed for further elaboration and feedback. A summary of his evaluation is provided below.
Usability and user friendliness was evaluated on the grounds of effectiveness, inexpensiveness and satisfaction. Effectiveness was defined as a way of measuring how accurately and completely the user goals would be achieved using Switch. Inexpensiveness was a measurement of the effort to achieve those goals. Satisfaction was defined the user’s own perception of the product.

The general impression of the application is very good. There were no major implications when it came to creating presentations, and the property grid provided a simpler (read: than PowerPoint) way of accessing attributes. He considered it easy to vary and design the slides and the overall impression was quite aesthetical.

A key idea in evaluation is to define the user groups. The current layout would probably work well for technically adept users, but might be difficult for others. The stimulus field in Switch should thus be reduced and provide more grouping of controls that are related. A concrete example is the small information box on the bottom of the property grid. The help text is not near the currently selected attribute and could thus easily be overlooked. Alternatives could be to increase the font, provide more distinct colors, or put it nearer to the grid elements. Another example is the sheer amount of buttons and elements in the interface. The more elements in the user interface, the more difficult it is to learn for new users. Furthermore, when people don’t understand something, they have a tendency to give up easily.

Reducing the complexity can be done several ways. One way is simply to reduce the number of user interface elements. This would mean a thorough walkthrough of the application to find out what elements can be removed. Another way is to highlight elements. By using colors and layout, it’s possible to relate similar families of functions better. Solbakken calls this the paradox of adding more information to reduce complexity. An example of this is button tool tips (tool tips were implemented the day before the usability tests and after the interview with Solbakken) or aggregators as described in chapter 6. The addition of more features in future versions will be a great challenge because of the above mentioned complexity in the interface. The users should also have the possibility to customize and tailor the user interface to their likings.

He finally requested more slide layout templates.
8. The Development of the Switch SWF Library

Up to this point mostly subjects related to the front end usage of the application have been discussed. However, a fairly long part of the development process has also been the creation of the SWF\(^1\) library. This library is what enables the user to transform a Switch presentation into the flash (SWF) file format played by browsers or a standalone flash player.

8.1 About the SWF File Format

The flash file format is vector based. This means that the colors and lines in the output render are not determined by colors assigned to individual pixels, but rather by mathematical vectors. Vectors can for example describe how a line of a certain width and with a certain color can go from one coordinate to another whereas conventional raster graphic would describe the colors of the pixels along the same path. This has a huge implication on zooming and is perhaps best explained by using a figure (figure 49).

![Figure 49: Difference between vector graphics and raster graphics. In the original vector image (a) a small piece has been cut out and scaled (marked in red). Figures b and c show this piece scaled 8x as vector graphics and raster graphics respectively.](image)

Since vectors are described mathematically, they enable infinite zooming and scaling without degradation. Raster graphics will on the other hand instead scale the individual pixels rendering crude looks. A downside to vectors is that they usually aren’t ideal for

\(^1\) On the contrary to many common beliefs, SWF is not an acronym. Macromedia hasn’t published that it stands for anything. It’s been widely proposed that it stands for Shockwave Flash or Small Web Format, but this is thus not the case. They do state, however, that the correct pronunciation is “\textit{swiff}” \cite{52}. 

photorealistic images or images with noise. This is because the more complicated the image is the more vectors are needed to describe it, so there are pros and cons to both formats.

Macromedia has developed the flash file format. The first editions had one main goal: to create compact files for displaying entertaining animations. One central idea was that the file format could be reused by players running on any system and that it would function on slow and unpredictable networks. Another goal was to design it to render at a high quality very fast [54]. Newer versions of the format also support audio, video and embedded raster graphics [53].

8.2 Elaborate Bytes

In order to understand some of the problems encountered during the library development process, a little parallel understanding of the file format is required. It’s no secret that the format is of high complexity and the library has thus taken lots of work and months to develop. As of today, the library is not yet complete and is not able to produce files with all the features of the flash file format. It wasn’t either planned to be completed since enabling all features would demand yet more months of person hours. It might be interesting to note, however, that a similar library written in the C# programming language has, as far as searching on the Internet has shown, not yet been developed.

The SWF library is a compiled DLL file with the sole purpose of producing SWF files. This is similar to other programming libraries and makes for easy reuse in future projects. As soon as the library is linked to a project, the project can take full advantage of all properties and methods in the library. This will for example enable the library to be linked to future web-projects to produce on-the-fly flash files with dynamic content.

A flash file consists of a series of tagged data blocks following a header and followed by an end tag (figure 50).

![Figure 50: SWF file structure](image-url)
This is the trivial part since all tags are byte-aligned and written separately to the file. A tag has itself a header and thus enables the player to skip it if it contains information the player does not understand.

There are two kinds of tags, definition tags and control tags. Definition tags define the content of the SWF movie – the shapes, text, bitmaps, sounds, and so on. Each definition tag contains a unique character ID, and the flash player stores the character\(^2\) in a repository called the dictionary. Control tags create and manipulate rendered instances of characters in the dictionary, and controls the flow of the movie [54]. To put something programmatically on screen in a flash movie one would thus first have to create it using a definition tag, and then place it somewhere using a control tag.

When it comes to creating the content of the tags, that’s where things get tricky. As already mentioned, the SWF file format is extremely compact. To save as many bits as possible, the file format goes to the extent of redefining literally every type\(^3\) in common programming languages. This has huge implications on the creation of the files. First of all, the sizes of the types vary dynamically. Depending on the actual values of the types, bits are removed so that the absolute minimum of bits is needed to describe the values. This is illustrated in the following minimal example: Suppose the (unsigned) integers 1, 2, 3 and 4 are going to be written in a tag. In the “normal” programming world, the integers are described using 32 bits each, namely 0...0001, 0...0010, 0...0011 and 0...0100. However, as seen in this example, only the last bits in each integer are actually describing the numbers. Flash would in this case rather indicate that an unsigned integer in this structure has length 3 (the longest bit-length of the four integers) and store the values as 001, 010, 011 and 100 respectively. Hence, only 12 bits plus the bit-length indicator are needed as opposed to 128 bits in the uncompressed case. This leads naturally to the second big challenge. Since values can be of an arbitrary bit-length, they’re not byte-aligned. The values have to be programmatically chopped up, bit-shifted, and packed one bit after the other, and then padded with zeros at the end of each structure that actually is byte-aligned. This

---

\(^2\) A character in this context refers to the full shape of something – a figure, a letter in a certain font and of a certain size, and so on.

\(^3\) A type is an entity in programming, such as an integer that is capable of holding positive and negative natural numbers between –2\(^{31}\) to 2\(^{31}\).
automatically implies that if a single bit is mistakenly written extra somewhere, the whole range of bits in the file gets shifted and the file becomes completely corrupt and unusable.

All this juggling of bits introduces a third problem: speed. Since structures are small and a file typically consists of tens of thousands of values, these bit operations need to work extremely fast. Every lag can in the worst case be multiplied thousands of times because of all the values written. Luckily, using knowledge of how the underlying hardware works, there are ways to speed this process up.

A common characteristic of the SWF library is that hardly any multiplications or divisions have been used. These are typically slow operations for a processor. In many cases these operations can be simplified as the numbers that are to be multiplied or divided aren’t really arbitrary. Instead, the library extensively uses bit-shift operations (similar to multiplying or dividing by two raised to a number), and AND and OR Boolean operations. Another optimization involves keeping and manipulating the tags in memory and writing the tags to file as a whole.

Since the structures that are used are so different, part of the problem involves efficiently translating between C# structures to structures used in flash. This problem is for example encountered every time a floating point value is being written to the file. In C# and according to the IEEE\(^4\) standards, floating point values are stored in a sign-magnitude form with a sign-bit, exponent and mantissa (figure 51).

![IEEE floating point number standard](image)

**Figure 51: IEEE floating point number standard.**

In SWF, however, a completely different format is used. First comes a 2 to 16 bit long field that represents the integer part of the number. Then follows a 16 bit long field that represents the decimal part of the number. An implication of this is hence that flash can not represent

---

\(^4\) Institute of Electrical and Electronics Engineers – one of the leading standards-making organizations in the world.
numbers closer to zero than $2^{-16}$, but that is not really important since moving or resizing in so small steps will not be visible. As already mentioned, also colors, gradients, and more or less every other structure need special translation before they are written to the file.

Now, one thing is translating between known structures. A completely other thing is that the flash file format doesn’t even define all common entities. For example, roughly speaking, a shape in flash consists of lines, curves and fills. Curves are always described as quadratic Bezier curves which are mathematically defined with two anchor points (beginning and endpoint) and a control point. The control point “bends” the line using a mathematical formula (figure 52).

![Figure 52: Quadratic Bezier curve with two anchor points ($P_1$ and $P_2$) and a control point ($C$).](image)

There are no circles, sine-curves, polynomial curves, or any other kind of structures that might aid drawing routines. In fact, accurate circles and ellipses cannot even be drawn. They have to be approximated using intricate mathematics by carefully calculated Bezier curves [55]. An example of this in practice is illustrated in figure 53.

![Figure 53: Approximating an ellipse using eight Bezier curves.](image)

And it gets worse. Also text and fonts are defined as shapes. This means that for every character (letter) that is to be included in the file, a routine has to analyze the font, pick it
apart, translate every curve in it into a quadratic Bezier curve, translate every straight line in the font into a straight line in flash, and at the same time make sure everything is scaled and positioned correctly, and that the endpoints of all lines and curves are connected. As if this wasn’t enough, other routines must ensure character code tables are correct (SWF uses UCS-2 encoding), and if necessary translate characters to Unicode. To not complicate this chapter overly much with details, it suffers to say that embedded bitmaps, jpeg tables, mp3 sound frames etc does not make things simpler.

To aid in this process, a testing and analysis program was developed. This program has been invaluable help in the library development process. The output files had to be analyzed for errors down to the very last bit and the only possible way to do this was to read the hexadecimal values themselves. Figure 54 shows a screenshot of this program.

![Figure 54: The SWF test program.](image)
9. Summary, Conclusions and Future Work

This project has looked at the process of prototyping an intuitive user interface for a presentation program. This was achieved by going through several stages in a research and development process.

First of all, the research problem was defined. Even though the core of the problem was to develop the interface for the presentation program, the problem also involved analyzing the user friendliness of the interface and determining the criteria for evaluating its usability.

Second, background information needed to be gathered. This has been an ongoing process throughout the research and has hopefully led to the outlines of a usable user interface. Just as important, it has led to a deeper understanding of what the different aspects of usability are and how they can be evaluated.

The development process was two-fold. First of all the interface was developed from scratch. This involved learning a whole new field of programming in the Windows environment. Only a few of the components in use in the interface actually exhibit standard looks and behavior so a lot of implementation work was needed. The programming efforts also spanned learning obscure Win API calls to achieve the more unusual and desired behavior from some of the components. Even though this was tedious and complicated work, it was extremely rewarding to learn a whole new field of programming. The second part of the development phase included the development of the SWF library. Although this was still programming in the C# programming language, the nature of this development proved to be very different. It involved studying in greater detail the structure of one of the most versatile and compact animation formats available and at the same time building something very useful. To complete the task, knowledge about complicated mathematics, hardware, vector graphics, font architecture, character encodings and more was needed. Not all this information was easily available, but in the process of finding information many things were learned. The programming phase has been long and hard, and an estimated 100.000 to 150.000 lines of code has been written. More importantly, it has been extremely interesting and very fruitful for future work.
Prototyping involves programming intertwined with usability testing, and in this case employing qualitative research. At that point a full set of scenarios and tests were needed for the testing phase and lots of useful insights were gathered from a day with test candidates in the lab. The tests also led to important insight in possible refinements of the interface, refinements that are already being worked at for future versions of the program. User characteristics from different guidelines turned out to be remarkably correct, yet the unorthodox approach of using a property grid to edit objects proved to be usable and intuitive. The iterative process will indeed continue and hopefully yield a user interface that is as good as possibly achievable.

9.1 Future Work

The continuation of this project will start where the thesis ends. Usability testing has already outlined possible refinements and there are still many features that have not been implemented. The SWF library still lacks many of the advanced features, and future possibilities in the program mean more usability testing and more iterations.

Many advanced features were not implemented because there simply wasn’t time to do so. Due to the ambitious nature of this project, the line had to be drawn somewhere. Improvements that might or should appear in next versions are:

- File association: Enables the user to double click a file in Windows Explorer and it will then be opened directly in Switch.
- File preview: A handy feature similar to the one supported by image files. When a file is selected in Windows Explorer, a bitmap preview of the file is shown in the left hand column. This enables the users to inspect a file in less than a second instead of opening Switch.
- Excel box: Creating a simple grid in order to display data and graphs.
- Real clipboard support: The clipboard used today simply buffers the data internally. Using the real clipboard, the data can easily be exported and new data imported.
- Possibility to import PowerPoint files.
- Possibility to import flash files and animations. This requires a whole new editor.
- Define rulers and a grid. This will enable objects to “snap” into place rather than giving the users freedom to move the objects anywhere. The feature can of course be turned on and off.
• More hot-tracking of controls. Highlight controls that are selected and mark the controls the mouse hovers over.
• The slides list was initially intended to look as smooth and slick as the Macintosh Dock. Since this cosmetic behavior wasn’t prioritized, it was never completed.
• Support of drag and drop, also from other products such as enabling the user to drag a slide from PowerPoint directly to Switch.
• Animations.
• Slide templates.
• Help functions
  o For a part of a program that is active, i.e. dialog boxes.
  o Selected object
    ▪ Using F1
    ▪ Balloon help
    ▪ Context-sensitive help
  o Prompts for fill-in fields
• Shake window to mark illegal actions
• Give all attributes in the property grid readable names.
• Produce more custom UITypeEditors.

Furthermore, Microsoft has developed something they call result oriented design. It resembles activity based planning because it shifts focus away from the fact that the users have to explore the interface to locate the functions and formats that enables them to do what they wanted. This is perhaps best illustrated with a diagram (figure 55):

![Diagram](image)

*Figure 55: An example of result-oriented design in practice.*

In the above figure, the user first creates a list. Then, by using a menu, button, or right-click in the list, the user can transform the list into a diagram. There should be several diagram types available, like arrow-based diagrams and Venn-diagrams. Finally, after a diagram type has been chosen, the user can chose a diagram style. The diagram style will determine the looks, colors and fonts of the diagram. Conveying these choices in a usable manner will prove a challenge.
As already outlined in this thesis, there is also room for quantitative analysis. Beta versions of a complete program will most probably be available on the Internet in about six months. From there, a special version of the program should be made that either tracks and reports bugs in the software, or that somehow gathers user feedback. By giving the beta versions away for free, the tracking systems can return much useful statistical information. Online questionnaires are another approach.

Another possible development for future versions might be to include customization and tailoring. This is where the property grid might come in handy again. Not only can it be used to display attributes and values for design elements, but one can imagine it being used to display selected attributes and values for interface elements. One can picture setting the program in a “customization mode” and thus give the user access to all these features. The code framework also makes this possible. Implementing and testing this kind of tailoring and customization could also prove to be an interesting thesis for future students.

9.2 Recap

A few days before the deadline of this project, searching on the Internet revealed a glimpse of the next generation Microsoft Office 2007 product line. As it turns out, they have redesigned the entire product interfaces from scratch, among other things throwing out all the toolbars as we know them. Instead, they have invented something called “The Ribbon”, and this is where things get interesting (figure 56).

![Figure 56: “The Ribbon” in Microsoft Word 2007.](image)

“The Ribbon” is a small panel that sits on the top of the application where the toolbars previously were. This panel is equipped with what Microsoft calls contextual tabs, meaning tabs with options and properties that change content depending on the context. It turns out context in this situation refers to the currently selected object, like a table or a block of text. Apparently Microsoft has identified the same usefulness for this type of control as was one
of the initial research hypotheses in Switch, namely the usage of the property grid. The contextual tabs take the control a bit further, though. One gets the impression that the properties dialogs are more or less a thing of the past. Instead, the contextual tabs are populated with i.e. styles that match the currently selected object, and simply moving the mouse over the image of the style changes the style of the object in the interface. This enables a preview that is a lot more efficient than the previous method. Microsoft has called this feature *Live Preview* and it appears to be a very handy one.

This study has been a long and laborious one. It was started out as an ambitious two-programmer project, and turned twice as big when the other programmer was forced to step down after only a few months. It is hence gratifying to learn that a big company such as Microsoft has decided that the world is ready for new elements in user interface. This could lead to both the demise of Switch even before it is completed or as breaking ground providing an easier acceptance for the somewhat unusual interface in Switch. The project has nevertheless been extremely interesting and has been worth every minute put into it.
10. Final Words…

"To design something really well you have to get it. You have to really grok what it's all about. It takes a passionate commitment to thoroughly understand something -- chew it up, not just quickly swallow it. Most people don't take the time to do that. Creativity is just connecting things. When you ask a creative person how they did something, they may feel a little guilty because they didn't really do it, they just saw something. It seemed obvious to them after awhile. That's because they were able to connect experiences they've had and synthesize new things. And the reason they were able to do that was that they've had more experiences or have thought more about their experiences than other people have. Unfortunately, that's too rare a commodity. A lot of people in our industry haven't had very diverse experiences. They don't have enough dots to connect, and they end up with very linear solutions, without a broad perspective on the problem. The broader one's understanding of the human experience, the better designs we will have.” (Steve Jobs, 1996) [8]
Citations


[2] Infoworld survey (June 12, 1995)


[64] Autodesk Media and Entertainment Solutions. 
http://usa.autodesk.com/adsk/servlet/index?siteID=123112&id=6397712


Of Windows Forms Apps. MSDN Magazine. 
http://msdn.microsoft.com/msdnmag/issues/06/03/WindowsFormsPerformance/

http://www.mmaonline.net/publications/MNMed2006/June/perspective%20-
%20segovis.html


http://arstechnica.com/articles/paedia/gui.ars

http://www.invisiblerevolution.net/engelbart/full_62_paper_augm_hum_int.html

[71] User interface, Wikipedia. 

[72] History of the graphical user interface, Wikipedia. 

Human-Computer Interaction. Interactions Magazine.


[75] Christopher Owen, Stephen Jang, Christopher Skeen, Mitchell Andreani. 
The History of Graphical User Interfaces.

[76] Eric Steven Raymond, Rob W. Landley. April 2004. The first GUIs. History: A Brief 
History of User Interfaces - Chapter 2. 
http://www.catb.org/~esr/writings/taouu/html/ch02s05.html

http://stevensonconsulting.com/index.php?option=content&task=view&id=50&Itemi d=2

[78] Microsoft PowerPoint, Wikipedia. 
http://en.wikipedia.org/wiki/Microsoft_PowerPoint


[85] Qualitative vs Quantitative analysis
http://www.ling.lancs.ac.uk/monkey/ihe/linguistics/corpus3/3qual.htm


[95] Usability, Wikipedia.  
http://en.wikipedia.org/wiki/Usability

http://hcibib.org/tcuid/chap-1.html


http://www3.sympatico.ca/maury/games/space/spacewar.html

http://www.jnd.org/dn.mss/affordance_conv.html

http://www2.sims.berkeley.edu/courses/is213/s01/lectures/Lecture5.ppt
Appendix A: 
Warm-up Questions for the Usability Tests

Name:
Age:
Occupation:

Have you ever used any presentation programs like e.g. PowerPoint?
If yes, which program?
If yes, on a scale from 0 to 10, how good would you range yourself?
If no, do you know what PowerPoint and slideshow presentation programs are?

Have you ever used any imaging programs like e.g. Adobe Photoshop?
If yes, which programs?
If yes, on a scale from 0 to 10, how good would you range your knowledge about graphics packages?

Have you ever used Macromedia Flash?
If yes, on a scale from 0 to 10, how good would you range yourself?
If no, do you know what Macromedia Flash is?

Describe your experience with computers and software.

A summary of the test participants and their skill levels is found in table 11.

<table>
<thead>
<tr>
<th>Participant</th>
<th>PowerPoint Experience (0-10)</th>
<th>Graphics Programs Experience (0-10)</th>
<th>Flash Experience (0-10)</th>
<th>User Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>Novice</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Novice</td>
</tr>
<tr>
<td>3</td>
<td>1-2</td>
<td>0</td>
<td>0</td>
<td>Novice</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>Advanced</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>6</td>
<td>4</td>
<td>Advanced</td>
</tr>
</tbody>
</table>

*Table 11: The test participants and their skill levels.*
Appendix B: Usability Test Transcripts

<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>Hhhhh</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 recent 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 is 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>

Table 12: Transcription symbols [97] [98].

10.1 Participant 1

R: Åpne programmet.
<Åpner programmet>
  Jeg tar opp disse menyene her jeg (.) så er de klare.
R: Da skal du velge en bakgrunn på selve presentasjonen. Det kan enten være en farge eller et bilde.
  Velger en farge her (.) på global.
<Klikker på bakgrunnsfargen for slideshowet>
R: Den samme siden skal nå ikke arve egenskapene fra slideshowet.
<Studerer properties’ene. Åpner bakgrunnsbilde-menyen for sliden. Åpner bakgrunnsfarge- menyen for sliden.>
  Skal jeg da (2) jeg har jo ikke laget noe slideshow enda. Her kan jeg jo forandre på den enkelte...
R: Slå av at den arver egenskapene til den globale
  Slå av de egenskapene at den er globale, ja...
<Klikker mange steder i property grid’et. Ser på attributenes navn.>
(10)
R: Slå av at den arver globale verdier. Med andre ord at hver slide kan ha et unikt bakgrunnsbilde.
(5)
Jeg går jo ikke på global her...

R: Poenget er at den ikke skal ha de globale settingene. De skal slås av for den enkelte sliden.

<Slår av arven>

R: Riktig

Den var litt vanskelig den derre der. Da vi gikk gjennom dette i sted hadde vi jo fire slider. Da kunne vi klikke på den enkelte sliden. Nå var det bare den første..

R: Da skal du legge inn en tittel i tekstboksen der.

<Dobbeltklikker i tekstboksen. Legger inn en tittel.>

R: Deretter skal du legge inn at den tittelen får en egen font.

<Markerer teksten> ((som ikke har noen hensikt)) <Endrer fonten>

R: Deretter skal du endre tekstboksens bakgrunnsfarge

(10)

<Endrer bakgrunnsfargen til tekstboksen>

R: Deretter skal du gi en egen farge og bredde til rammen

<Endrer bredde og farge på rammen>

Slik

R: Legg til en ny slide

<Lagger til en ny slide ved å bruke menyen>

R: Velg den nye sliden og sett et nytt bakgrunnsbilde på slideshowet

<Velger den nye sliden>

R: ...på hele presentasjonen

...på hele presentasjonen, ja.

Da går jeg her, på global.

<Velger bakgrunnsbilde på slideshowet>

R: Sett inn en tekstboks i slide 2

En tekstboks til?

R: Ja. Du kan godt slette en av de som er der hvis du vil.

<Sletter en tekstboks vha kontekst-menyen>

<Sletter inn en ny tekstboks ved å klikke en gang på work-area’et. Justerer deretter posisjon og størrelse på eget initiativ> ((Brukeren burde ha klikket og dratt for å sette inn tekstboksen))

R: Slett slide nummer 2

<Sletter slide nummer 2 ved å bruke kontekst menyen og "Cut”>

R: Lagre slideshowet

Lagre?

R: Lagre, ja.

<Lagrer slideshowet>

R: Lag en forhåndsvisning av sliden

<Forhåndsviser sliden> ((Det dukker opp en error-boks som brukeren klikker bort uten å lese...))

R: Til slutt kan du åpne den filen du nettopp har laget ved å bruke Windows Explorer. Slik som du gjorde i sted ved å navigere til...

<Åpner Windows Explorer og navigerer seg frem til filen. Åpner filen>
10.2 Participant 2

R: Åpne programmet.
<Åpner programmet>
R: Så setter vi en bakgrunnsfarge på sliden.
<Setter bakgrunnsfarge på sliden>
   Den tok ikke nå...
   <Klikker på property grid’et en gang til> ((Åpnenbart forvirret over at bakgrunnsfargen på
   sliden ikke endret seg nå som inheritance var slått på))
   R: Sånn ja. Du må jo slå av arv for at det skal synes.
   <Klikker litt på grid’et og slår av arv>
   R: Så endrer du teksten ((...i en tekstboks))
   unselect’es. Klikker nesten ”delete”, men tar seg i det siden hun er redd for at hele boksen
   slettes.>
      Cursor har du ikke... ((boksen er verken valgt eller hun har dobbeltklikket på den))
   <Klikker på boksen og utenfor noen ganger. Dobbeltklikker til slutt på boksen og ser
   cursor’en. Skriver.>
   R: Deretter skal du endre fonten til noe litt større.
   <Klikker på farge-menyen på verktøylinjen>
      Nei, der var det ikke...
   <Klikker på f ontstørrelsen på verktøylinjen og endrer den.>  
      Ikke den der...
   <Flytter på tekstboksen ved en feiltagelse>
      Jeg blander med sånn vi gjorde før. Den svartnet og så forandret man teksten. Det kan
      man ikke nå her...
   R: Du har gjort det en del ganger allerede. Husk hva jeg sa da. Forskjellen på at den der
      (verktøylinjen) gjelder n y e ting.
      Nei...jeg finner den ikke jeg...Her er den jo...
   <Klikker på font-størrelsen på verktøylinjen. Endrer den der.>
   R: Sånn ja.
   R: Deretter legger du til en ny slide.
      En ny firkant?
   R: En ny s l i d e
   <Dobbelt-høyreklikker på slides-listen. Legger til en ny slide.>
   R: Deretter skal du endre den globale fargen.
   <Leter i property grid’et der tekstboksen nå er selected>
      Det er slide appearance det...
   <Leter mer. Klikker på tekstfargen.>
      Nei, det er ikke den. Det går på tekstfargen det, ikke på bakgrunnen. Gjør det ikke det?
      (.) Appearance...
   R: G l o b a l e settinger...
      Jeg finner ikke det ordet jeg...
   R: Kanskje fordi du har trykket på feil...
   <Leter mer i property grid’et>
   R: Trykk på slide’n.
      Der er den ja
   <Endrer den globale bakgrunnsfargen>
   R: Deretter endrer du teksten på slide nummer to.
R: Så kan du endre fonten i den tekstboksen der.
Fonten? Hva var det igjen da? (2) Font ja...
R: Tekst-utseendet kan man kalle det.
R: Så sletter du slide nummer to.
<Dobbelt-høyreklikker i slides listen. Bruker cut til sletting.>
R: Deretter lagrer du.
<Lagrer ved å bruke fil-menyen.>
R: Så tar du en forhåndsvisning, preview slide (2) knappen nederst. ((så at testpersonen ikke husket hva en forhåndsvisning var))
<Klikker på forhåndsvisningen>
R: Ok, lukk den. Så kan du finne igjen filen ved å gå på ”my computer” i filhierarkiet.
<Klikker vekk forhåndsvisningen. Roter litt med windows explorer>
Jeg finner den ikke jeg.
R: Dobbeltklikk på...
<Dobbeltklikker riktig>
Her ja... (3) dobbeltklikke på den og? <henviser til katalogen>
<Finner filen. Åpner den.> 

10.3 Participant 3

R: Da kan du starte med å åpne programmet.
<Åpner programmet>
<Velger bakgrunnsfarge på siden. Slår av inheritance.>
Det er vel bare overskriften, er det ikke det? ((Henviser til kategori-navnet))
<Dobbeltklikker der det står ”text” i property grid’et. Stusser litt over at det ikke skjer noe.>
R: Tenk hoyt fortsatt. Hvorfor dobbeltklikker du på ”text” for eksempel.
Jeg dobbeltklikket vel i k k e.
R: Hva står det i boksen da?
Click here (2) to add title.
<Dobbeltklikker i tekstboksen. Skriver inn tekst.>
R: Neste punkt er at fonten, typen, i denne boksen får en annen skrift eller farge. Gjør en endring på fonten etter eget ønske.
Forandre typen, for eksempel? Eller fargen?
<Velger font fra rullegardin menyen på verktøylinjen>
Det har jeg ikke gjort før.
<Studerer property grid’et i ca 10-15 sekunder>
R: Font er det samme som type, bare for å understreke det (”font” var et nytt begrep for testpersonen))
Å ja, font ja.
<Studerer property grid’et i noen sekunder til. Åpner font-dialogen>
Det var der hvor (3) ja da kan jeg jo velge noe annet da. Jeg aner jo ikke hva jeg velger.
R: Det er ikke så farlig.
<Velger en font>
R: Da går vi videre til neste punkt. Det er at du skal endre utseendet på tekstboksen du
akkurat har skrevet i. Det vil si at den skal få en bakgrunnsfarge, en farge på rammen og
en størrelse på rammen.
<Velger tekstboksen. Studerer property grid’et i ca 20 sekunder. Endrer tykkelsen på
rammen>
Der ja. Hva var det mer jeg skulle gjøre?
R: Du skal endre bakgrunnsfargen på boksen.
Den er jo nå hvit, ikke sant?
R: Ja.
Appearance...
<Studerer property grid’et noen sekunder til. Endree bakgrunnsfargen på boksen.>
R: Så skal du endre fargen på linjen.
Den?
<Peker på linjen med musen>
R: Ja.
Der var typene...
<Studerer property grid’et i ca 20 sekunder til.>
Jeg skulle...?
R: Endre fargen på linjen.
<Endrer fargen på linjen>
R: Den er god. Deretter legger du til en ny slide.
<Legger til en ny slide ved å høyreklikke på slides listen.>
R: Deretter velger du en setting på hele presentasjonen
  Begge to? ((slidene))
R: Ja, altså en global setting. Den gjelder jo for begge når den gjelder globalt.
  Hva skulle jeg gjøre? Endre farge?
R: Enten sette en farge eller et bilde på den globale settingen.
<Velger en bakgrunnsfarge på slideshow’et>
R: Deretter skal du endre teksten i tekstboksen i slide nummer to.
<Velger slide nummer to.>
  Der står det ”click here to (.) add text”. Jeg skal forandre teksten?
R: Du skal forandre teksten, ja.
<Klikker i ”text” feltet i proerty grid’et ((slik som sist)). Høyreklikker på tekstboksen ((slik
som sist)). Stopper noen sekunder, dobbeltklikker deretter på tekstboksen og endrer
teksten.>
R: Deretter skal du endre fonten, typen, enten i farge eller størrelse. På den samme
  tekstboksen.
<Endrer font fargen>
R: Deretter skal du sette inn en tredje slide.
<Setter inn en slide ved å høyreklikke på slides listen.>
R: Så kan du slette densliden som er i midten, slide nummer to.
<Sletter sliden ved å høyreklikke på den>
R: Så lagrer du.
  Hele?
R: Ja. Det går ikke å bare lagre en slide.
<Høyreklikker på en slide. Studerer alle menyvalgene i ca 20 sekunder.>
Hva var det ”paste” betyr?
R: Det trenger du ikke bry deg om. Du skal la große presentasjonen. Da skal jeg opp på ”fil” her.
Lager presentasjonen.> 
R: Deretter skal du lage denne filen som du åpnet i det andre programmet. Det kunne du for eksempel gjøre ved å trykke for å få opp denne visningen i et eget vindu. Hvis du husker hvordan vi gjorde det i stad. Ehm...
R: Vet du hva jeg snakker om?
Ja vi skulle få det i sånn (.) få det i stor størrelse kanskje og?
R: Ja i alle fall få det opp i et eget vindu. På ”fil” det kanskje...
R: ”Preview slide” funksjonen.
<Klikker på filmenyen. Leter litt. Klikker så på ”preview slide” knappen.>
R: Så skal vi finne denne filen. For den er blitt lagret inni filsystemet i datamaskinen. Det har vi da at vi fant frem til denne lokasjonen der denne finnes. Det gjøres ved å trykke på ”My Computer”.
<Åpner ”My Computer”. Leter.>
R: Lokasjonen står på tavlen.
<Enkeltklikker på C:\. >Vi skulle...?
R: Finne frem filen i filsystemet.
<Trykker på ”file” i Windows Explorer.>
Nei...
<Klikker på C:\ igjen. Så en gang til. Windows Explorer endrer modus til å kunne redigere navnet på C:\>
R: For å åpne den må du dobbeltklikke.
<Dobbeltklikker.>
R: Men ikke nå for nå er den satt til å endre navnet.
<Klikker utenfor. Dobbeltklikker deretter på C\. Leter i ca 30-40 sekunder blant filene.>
R: Hvis du ser på tavla så ser du filnavnet og hvor den befinner seg.
<Prover å finne C:\ igjen>
R: C:\ har du åpnet. Neste nivå er...?
    Tmp. T m p. (4) Jeg ser ikke hvor det står t m p her jeg. (4) Der? ((henviser til katalogen temp)).
R: To ned. Der ja.
<Åpner C\tmp. Studerer innholdet i katalogen, som er én fil.>
R: Filen heter...? (. ) Det står også på tavla.
<Enkeltklikker to ganger på filen slik at Windows Explorer settes i modus slik at navnet på filen kan endres.>
Ja det er den.
R: Ja, men nå satte du den i modus for å endre navnet igjen.
<Klikker utenfor. Dobbeltklikker på filen og åpner den dermed i Flash Player.>
R: Til slutt skal du sette den i fullskjerm modus.
.Setter den i fullskjerm modus.>
10.4 Participant 4

R: Da velger du en bakgrunn på selve sliden.
   Selve sliden? Mener du da slideshowet eller den der ((peker på sliden med musen))
R: Selve slideshowet
   <Velger en bakgrunn på sliden. Slår av arv.>
   Det som er problemet er menyen ((peker på property grid’et)). Det tok meg litt tid å
   komme meg inn i det. Hvis du får litt mer luft inn i det så blir det enklere å bruke.
   Bokstavene blir litt presset sammen og så forsvinner de under her ((henviser til at de
   lange variabelnavnene (som skal endres) ikke alltid får plass i høyre kolonne)). Bruk
   kanskje knapper eller noe.
R: Så kan du slette de to tekstboksene.
   <Sletter tekstboksne ved å bruke delete-knappen>
R: Så skal du sette inn en ny tekstboks.
   <Setter inn en ny tekstboks ved hjelp av klikk-og-dra.>
R: Deretter setter du inn en eller annen tekst i tekstboksen
   <Dobbeltklikker i tekstboksen. Setter inn noe tekst.>
R: Deretter endrer du fonten.
   <Markerer teksten. ((hvilket er unødvendig, men ikke har innvirkning)). Endrer fonten.>
   Det gjør ikke noe at alt er her og ikke der ((henviser til property grid’et)).
R: Sett inn en ny slide.
   Ny slide!
   <Setter inn en ny slide ved å trykke på knappen i verktøylinjen.>
R: Deretter velger du en slideshow bakgrunn.
   <Velger en slideshow bakgrunn>
R: Lag en ny slide.
   <Lager en ny slide ved å trykke på knappen i verktøylinjen.>
R: Deretter sletter du begge tekstboksene.
   <Sletter tekstboksene ved å velge cut i kontekstmenyen>
R: Så kan du tegne en stjerne.
   <Tegner en stjerne>
R: Deretter skal du lage en swf-fil. En flash fil.
   <Klikker i fil-menyen>
R: Publish
   <Publiserer>
R: Da er den filen lagret i katalogen som står der. Da kan du gå til C:\tmp og åpne filen der,
   så kjører du den i fullscreen.
   <Åpner filen. Kjører den i fullscreen.>

10.5 Participant 5

R: Først kan du velge en bakgrunnsfarge eller et bilde på en enkelt slide.
   <Velger en bakgrunnsfarge. Slår av arv.>
R: Deretter skal du sette inn en tekstboks i sliden din
   <Setter inn en tekstboks ved å klikke og dra.>
R: Deretter endrer du tittelen i tekstboksen du akkurat lagde.
<Endrer teksten.>
R: Så endrer du bakgrunnsfargen, fargen på omrisset og bredden på omrisset på den samme tekstboksen.


<Endrer bakgrunnsfargen, linjefargen og linjebredden>
R: Så kan du sette inn en ny slide.

<Setter inn en ny slide ved hjelp av hurtigtastene.>
R: Så kan du endre bakgrunnsbildet eller bakgrunnsfargen for selve slideshow’et.

<Endrer bakgrunnsfargen til slideshow’et>
R: Deretter kan du tegne en stjerne ved hjelp av tegneverktøyet.

<Sletter de to tekstboksene som er i sliden som default. >

<Tegner en stjerne.>
R: Så lagrer du.

<Lagrer ved hjelp av knappen på verktøylinjen.>
R: Deretter så eksporterer du slideshowet.

Det er vel ”publish” det?
R: Ja.

<Publiserer slideshowet fra filmenyen.>
R: Til slutt så kan du gå til det directoryet der den filen ligger.

<Finner frem og åpner filen>
R: ...og åpne i fullscreen.

<Åpner i fullscreen>