Patterns as Tools for User Interface Design

Martijn van Welie, Gerrit C. van der Veer, Anton Eliëns

Vrije Universiteit, Department of Computer Science
De Boelelaan 1081a, 1081 HV Amsterdam, Holland
{martijn,gerrit,eliens}@cs.vu.nl

Abstract. Designing usable systems is difficult and designers need effective tools that are usable themselves. Effective design tools should be based on proven knowledge of design. Capturing knowledge about the successful design of usable systems is important for both novice and experienced designers and traditionally, this knowledge has largely been described in guidelines. However, guidelines have shown to have problems concerning selection, validity and applicability. Patterns have emerged as a possible solution to some of the problems from which guidelines suffer. Patterns focus on the context of a problem and solution thereby guiding the designer in using the design knowledge. Patterns for architecture or software engineering are not identical in structure and user interface design also requires its own structure for patterns, focusing on usability. This paper explores how patterns for user interface design must be structured in order to be effective and usable tools for designers. A structure for user interface design patterns is proposed and is illustrated with an example.

1 Introduction

Guidelines have since long been used to capture design knowledge and to help designers in using that knowledge when designing user interfaces. The design knowledge helps the designer to make the right design decisions and prevents the designer from making the same mistakes over and over again. However, applying guidelines is not without problems. Usually guidelines are numerous and it is difficult to select the guidelines that apply to a particular design problem. Additionally, guidelines may seem to contradict each other and consequently the designer may still not solve the design problem. Guidelines are usually very compact but their validity or appropriateness always depends on a *context*. Software tools for working with guidelines can help but do not address the core problems of guidelines. Instead of offering software tools for working with guidelines, we propose patterns as a solution to some of the problems of using guidelines. Patterns explicitly focus on context and tell the designer *when*, *how* and *why* the solution can be applied. Hence, patterns can be more powerful than guidelines as tools for designers. Inspired by the work of Alexander [1], patterns have become popular in software construction [6]. Interest in patterns for user interface design (UID) goes back to 1994 [2,9] but a proper set of such patterns still has not emerged. Some attempts have been made to create patterns but there appears to be a lack of consensus about how patterns for UID should be written down, which focus they should have and how they should be structured. Consequently, a potentially even

more interesting pattern language for UID has not been established since it is necessarily preceded by the development of a sufficiently large body of patterns. In Section 2, we will take a closer look at why patterns can be more effective than guidelines. In Sections 3 and 4 we will look at the definition of patterns and how that translates to patterns for UID. In Section 5 we will propose a template for UID patterns focused on usability and will discuss and illustrate the template with an elaborated example.

2 Guidelines or Patterns?

The purpose of guidelines is to capture design knowledge into small rules, which can then be used when constructing new user interfaces. A pattern is supposed to capture *proven* design knowledge and is described in terms of a problem, context and solution. Since they have more or less the same purpose, the format may seem the only difference. On one hand it is true that the design knowledge of a guideline could also be written down using a pattern template. On the other hand, the fact that a template is used to write down the guideline does not necessarily make it a pattern. For patterns it is important that the solution is a *proven* solution to the stated problem and the designers agree upon the fact that it *is* a proven solution. Designers share values and ideas so the pattern must relate to their experience. With guidelines this is often an issue because guidelines are usually not explained together with a rationale. In the Smith and Mosier guidelines [11] some guidelines have a short rationale in the comment field but they are often simply defined without any argumentation whereas some are just style definitions and not generic guidelines.

It has often been reported that guideline have a number of problems when used [4,8]. Some of the problems are:

- Guidelines are often too simplistic or too abstract
- Guidelines can be difficult to select
- Guidelines can be difficult to interpret
- Guidelines can be conflicting
- Guidelines often have authority issues concerning their validity

One of the reasons for these problems is the fact that most guidelines suggest a general absolute validity but in fact, they can only be applied in a specific *context*. This context is crucial for knowing which guidelines to use and why. For many design decisions, it is simply required to know the tasks of the users and the characteristics of the users. Without that knowledge, the design problem cannot be solved adequately. Guidelines have no intrinsic way of stating the context for which they apply and at most, it is briefly mentioned.

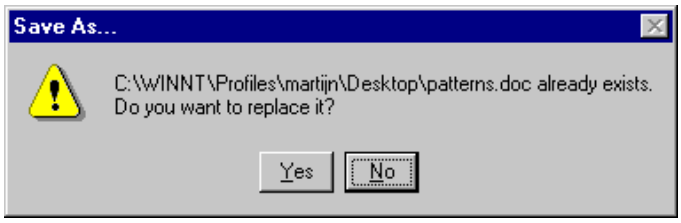
Another problem of guidelines is that it is often difficult to see what the *problem* is and why the guideline is like it is. For example, consider a very simple guideline saying *"Left align labels in dialog window"*. What is the real problem being addressed by this guideline? It is not "how to layout labels" because that would be a problem for the UI designer. But what is the benefit for the *end-user*? In our opinion, the real problem should be concerned with understanding information on a display with aspects such as scanning time and readability which goes back to Fitt's law [5]. A pattern makes both

the *context* and *problem* explicit and the *solution* is provided along with a *rationale*. Consequently, compared to guidelines, patterns contain more complex design knowledge and often several guidelines are integrated in one pattern.

Guidelines exist usually in two forms; do this or do not do this. Patterns focus on "do this" only and are hence prescriptive and *constructive*. Further more, solutions need to be very concrete and should not raise new questions surrounding the solution.

3 An Example

The following pattern is a very simple example of a pattern for user interface design. It is focused on the use of warning messages to protect the user.

Name	**The Shield**
Problem	**The user may accidentally select a function that has irreversible (side) effects.**
Usability Principle	Error Management
Context	The user needs to be protected against unintended or accidental actions that have irreversible (side) effects. The (side) effects may lead to unsafe or highly undesired situations. For example the unintended deletion or overwriting of files. Do not use for actions that are reversible.
Forces	- The user is striving for speed while trying to avoid mistakes. - The severity of the (side) effects.
Solutions	**Protect the user by inserting a shield.** Add an extra protection layer to the function to protect the user from making mistakes. The user is asked to confirm her intent with the default answer being the safe option.
Examples	 A copy of the file already exists at the specified location. Overwriting it will result in loss of the copy. The default is “No” so that the speedy user has to take the effort of saying “Yes”.
Usability Impact	Increased safety, less errors and higher satisfaction. However, it requires extra user action which leads to lower performance time.
Rationale	The extra layer causes the user to require 2 repetitive mistakes instead of 1. The safe default decreases the chances for a second mistake.
Known Uses	Microsoft Explorer, Apple Finder

The pattern is related to a problem that a user might have, how it can be solved and why it works. The pattern contains knowledge that would otherwise be described in at least two guidelines; "choose save defaults" and "ask for confirmation".

4 Patterns as Tools

Patterns are potentially better tools than guidelines because they explicitly are related to a *context* and are *problem* centered. Although this may conceptually be true, in practice creating patterns for UID is not that easy. A pattern for UID is not necessarily structured in the same way as an architecture pattern and it is important to find a format that has been designed for UID and has the right *view* on the important issues in UID. Suitability for describing usability related problems is an important issue for UID patterns. In this section, we will define pattern and propose a format for them.

4.1 Defining a Pattern

As the name pattern already suggest, a pattern is concerned with repeating elements, problems and solutions than emerge. Alexander [1] defines a pattern as follows; *"Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution"*. He goes on explaining the nature of a pattern; *"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over…"*. From these explanations it shows that patterns are very *practical*, they describe instances of "good" design and not vague principles or strategies. Further more, they have been proven and are hence not theories or speculation. It is therefore necessary that a pattern contains a rationale why the solution works and proof by referring to examples where the pattern was successfully applied. Patterns are *prescriptive* and help designers *construct* new instances. Alexander said a pattern should describe the *core* of a solution. Other related issues concerning the context are therefore dealt with by other related patterns that are being referenced to. Patterns for different purposes usually do not exactly have the same template and for each purpose an adaptation is needed. The main fields are always *problem*, *context*, *solution*, and *forces*. The remaining fields are extensions that should help make the knowledge even more clear.

4.2 Anti-Patterns

Within the Software Engineering community, the success of patterns led to the development of anti-patterns [3]. Anti-patterns focus on why things are not going right and then a solution is given. It can be seen as a pattern that is preceded by an example of bad design. It shows how not to do it and then how to solve it. Therefore, anti-patterns are *descriptive* and reflect on a particular design choice. In user interface design, many examples of bad design have been documented[1]. Seeing bad designs may be very inspiring but it does not directly help to solve problems.

Patterns and anti-patterns can also be combined by extending the normal pattern with an example of what is likely to happen if the pattern is not used. What is the dan-

[1] For example the "Interface Hall of Shame", http://www.iarchitect.com/shame.htm

ger of not solving the problem right? In particular for UID this may be very illustrative because the “danger” can often be shown with a single screenshot.

5 Writing Patterns for UID

Patterns for UID should share the same philosophy as patterns for architecture or software construction. However, the exact format for a pattern depends on the “topic” and therefore patterns for UID also need a specialized format. We can learn from SE patterns in the sense that those patterns also needed a modification. Patterns are written with a certain “view” on the problems. In architecture this was defined as “quality without a name”, a comfortable or enjoyable living environment. In SE the view is related to re-use, flexibility and efficiency of the system. In our opinion, the view for UID patterns should simply be *usability*. Patterns for UID should help making systems more usable for humans in the same way as Alexander's pattern made living more pleasant to humans. Therefore, before we can define a format for UID patterns we need to understand what usability is so that the important aspects of the format can be derived.

5.1 A View on Usability

Many different definitions of usability exist, making usability a confusing concept when actually designing a new system. Fig. 1 shows a layered model of usability [13] that helps understanding the concept of usability. On the highest level, the ISO definition of usability is given, split up in three aspects: efficiency, effectiveness and satisfaction. This level is a rather abstract way of looking at usability and is not directly applicable in practice. However, it does give three solid pillars for looking at usability that are based on a well-formed theory. The next level contains a number of usage indicators which are indicators of the usability level that can actually be observed in practice when users are at work. Each of these indicators contributes to the abstract aspects of the higher level. For instance, a low error-rate contributes to a better effectiveness and good performance speed indicates good efficiency. The desired "level" for each of the usage indicators depends on the nature of the system. For a production system efficiency may be the main goal, but for an entertainment website satisfaction may be far more important than efficiency.

One level lower is the level of means. Means cannot be observed in user tests and are not goals by themselves whereas indicators are observable goals. The means are used in "heuristics" for improving one or more of the usage indicators and are consequently not goals by themselves. For instance, consistency may have a positive effect on learnability and warnings may reduce errors. On the other hand, high adaptability may have a negative effect of memorability while having a positive effect of performance time. Each means can have a positive or negative effect on some of the indicators. The means need to be "used with care" and a designer should take care not to apply them automatically. The best usability results from an optimal use of the means where each means is at a certain "level", somewhere between "none" and "completely/everywhere/all the time". It is up to the designer to find those optimal levels

for each means. In order to do so the designer has to use the three knowledge domains (humans, design, and task) to determine the appropriate levels. For example, when design knowledge is consulted by using guidelines, it is clear that the guidelines should embody the knowledge of how changes in use of the means affect the usage indicators.

The list of usage indicators is complete in the sense that these ones have been identified in literature. The number of possible means however is quite large and only some examples of means are shown in Fig. 1. The means can be grouped according to the ergonomic principle that is involved. Scapin [10] suggests a categorization of such principles, which include *guidance*, *workload*, *explicit control*, *adaptability*, *error management*, *consistency*, *significance of codes* and *compatibility*.

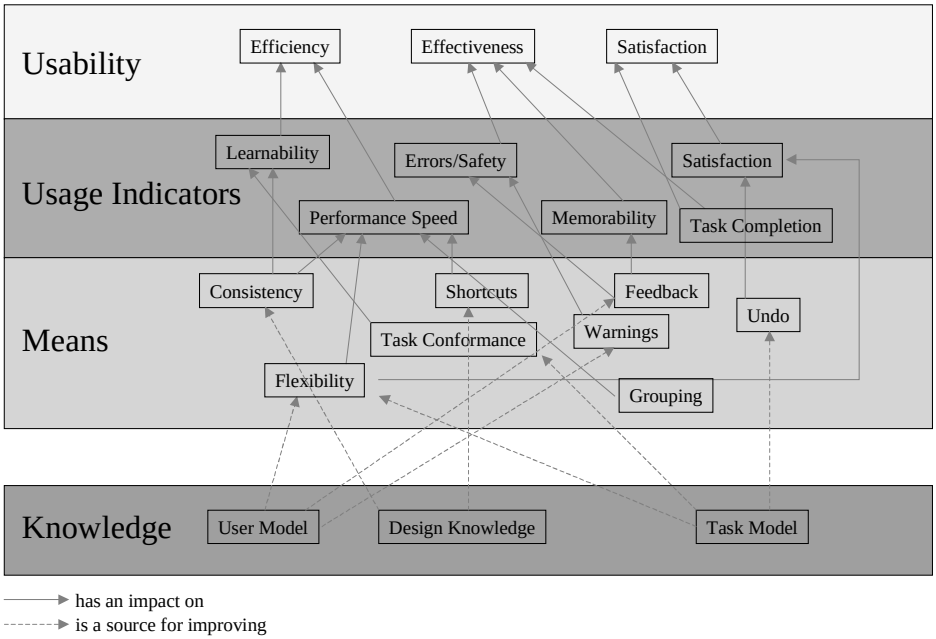


Fig. 1. A layered model of usability

5.2 A Template for Design Patterns

A pattern for UID should be focused on solutions that improve the usability of the system in use. From the usability model of the previous section we can see that improvements in usability must be measurable in usage indicators. Each pattern should therefore state the impact on these usage indicators. In short, if a UID pattern does not improve at least one usage indicator, it is *not* a UID pattern. Preferably, a pattern should be based on an ergonomic principle [10] such as *user guidance*, or *consistency*, or *error management*. The rationale section should explain how the ergonomic principles as used in the solution lead to an improvement of the usage indicators. In contrast, guidelines usually describe the usage of means without referring to the relevant usage indicators or context of use.

The main elements of each pattern can be used directly for UID patterns as well. However, it is important to write them down in the right "view".

- *Problem*. Problems in UID patterns should be usability problems of the system in use. Problems are related to usage of the system and are relevant to the user or any other stakeholder that is interested in usability. In contrast to SE patterns, problems in UID patterns should not be focused on constructional problems designers are facing. Hence, problem descriptions should often be user task oriented.
- *Context*. The context is also focused on the user. What are the characteristics of the context of use, including the tasks, users and environment for which the pattern can be applied?
- *Solution*. A solution must be described very concretely and must not impose new problems. However, a solution describes only the *core* of the solution and other patterns might be needed to solve sub-problems. Other patterns relevant to the solution should be referenced to.
- *Examples*. The example should show how the pattern has been used successfully in a system. An example can often be given using a screenshot and some additional text to explain the context of the particular solution. It is preferred to use examples for real-life systems so that the validity of the pattern is enforced. If a writer cannot find any real-life example, the pattern is either not a good pattern or rarely applied.

The fields and "view" needed to write UID patterns are important. For example if the view is taken wrongly, one might write patterns on "how to use tab controls". This is very tempting to do especially when guidelines are rewritten into pattern format. Such views take on the perspective of the designer and not the user. Moreover, the design knowledge about "how to use tab controls" depends on the context of when it is applied, the users and their task. In other words, it is looking at the problem from the point of the solution without knowing the problem. The example in the appendix shows the complete template with fields that are specific for UID patterns.

Sutcliffe has also proposed a way of describing the contents of a pattern using a claims approach [12]. Apart from terminology (Scenario for Example, Effect for Usability Impact etc.) the structure is very similar. However, there is neither an explicit problem statement nor a context specification.

5.3 An example: The Wizard Pattern

In the appendix the *Wizard* pattern is given. The wizard is a well-known artifact that can be found in many applications such as installation programs but also in ATM's. Many guidelines tell designers how to use the wizard or how to design effective wizards. Naturally it is good to use them when writing patterns. The first problem when writing a pattern for the wizard phenomenon, is thinking about what *exactly* the problem is for which the wizard is a solution. Characteristic is that it is concerned with a task that is a basic task to the user. The user thinks about the task as "one thing". For example, "install a program". However, the task has several subtasks where decisions need to be made and the wizard helps the user take these steps. Then the context needs to be defined and the exact constraints need to be formulated because the wizard is not

always *the* solution for the problem. The context puts the constraints such as the user expertise and the number of subtasks. The problem statement, context definition and forces are difficult to get right and the pattern writing community will certainly need several iterations. Then the solution needs to be described in a way that is as general as possible without describing a particular instance of the wizard. For example in this pattern we choose to speak about navigational widgets instead of a “next and previous button” because that is not the essence of the solution. It is the possibility to navigate in a sequence of tasks. The rationale and usability impact field then explain why this solution works for the problem in the specified context.

5.4 A Network of Patterns

Within Alexander's collection of patterns and also in the SE pattern collection, a network of patterns is used to connect patterns. Mahemof [7] has already suggested several different kinds of patterns for UID. Certain patterns can deal with small problems that deal with only one screen while others focus on a high level principle such as choosing for direct manipulation. In our opinion, such a hierarchy will appear only when a sufficient number of patterns have been identified so that distinctions can be made. The structure of a collection should be based on how the patterns are used in practice. Since it is premature to make assumptions about the actual usage of patterns, we will restrict the scope of our paper to the development of patterns.

6. Patterns Collections

Although interest in patterns for UID has existed for some years, patterns are still not widely available, let alone pattern collections. Currently there are two collections available. The first one was compiled by Jenifer Tidwell[2] and contains ± 60 patterns. The other collection was compiled by the Usability Group of the University of Brighton[3] and contains only a dozen patterns. When comparing the patterns from these collections, it is clear that there is a large difference in the format that is used. The Tidwell collection is structured using the standard fields; name, problem, context examples and forces. The Brighton collection is not so structured and used a narrative form filled with examples of “bad” design as introduction to the pattern. Both collections contain anecdotes that illustrate the pattern. However, closer inspection of the patterns shows that *writing* patterns is not a trivial task. Some patterns are just direct rewrites of guidelines. Other problems lay in the essence of the pattern; the problem description and the solution. Problems are often described very vaguely and the attention is quickly focused to the solution. However, this is contradictory to the purpose of a pattern. It is important that the problem is formulated accurately so that it can be verified that the solution is actually solving the problem. Consider the example of the "Status Display"[4] pattern:

---

[2] http://www.mit.edu/~jtidwell/common_ground.html
[3] http://www.it.bton.ac.uk/cil/usability/patterns/
[4] http://www.mit.edu/~jtidwell/language/status_display.html

- **Problem**: How can the artifact best show the state information to the user?
- **Solution**: Choose *well-designed* displays for the information to be shown. Put them together in a way that emphasizes the important things, de-emphasizes the trivial, doesn't hide or obscure anything, and prevents confusing one piece of information with another.

This pattern states a problem that is not directly a problem of the *end-user*. Moreover, the solution is rather vague and creates new questions such as what exactly is a "well-designed display"? A solution description should avoid phrases like "…design in a manner that emphasizes…", "…use self explaining labels for…", "…choose the most appropriate…" which all contains subjective judgment and consequently do not contribute to describing the *core* of a solution.

6.1 The Amsterdam Pattern Collection

The example pattern "The Wizard" in the appendix is one of the patterns that can be found in the collection we started[5]. At the time of writing, our collection contains twenty patterns that have been formulated using the template given in the previous section. The reason for starting a new collection is that we wanted a collection of patterns that is strictly focused on problems of the *end-user* and not problems of *designers*. The patterns are *candidates* since the process of reaching consensus is still progressing. Anyone can submit new patterns and the patterns are being discussed by a small group of researchers and practitioners. The site is built using XML in order to create a consistent and standard format for publishing patterns. Additionally, the use of XML facilitates several ways of automatic indexing or categorizing of patterns. Pattern writers can submit a pattern using the pattern DTD which causes all patterns to be rendered in a consistent way. Other patterns in our collection are "The Canonical Grid", "Dead or Alive", and "The Shield". We try the use metaphors in our pattern names in order to improve the development of a pattern language.

7. Towards a Pattern Language

When a community agrees upon a collection of patterns, it is possible to speak of a pattern *language*. Patterns are usually related to each other and consequently a network of patterns constitutes a pattern language. The development of a pattern language is the highest goal in pattern research. However, before we can speak of a pattern language for user interface design it is necessary to develop *good* patterns. In this paper we have outlined a format for UID patterns and illustrated the format with an example. We are now actively working on the development of a substantial amount of patterns and the evaluation of the patterns to create the very important *agreement*. Validity and agreement are requirements for a pattern language. Not anything written down in a pattern form is a pattern and should not be accepted as such. Up till now, our work has focused on the *development* of patterns but in the near future the pattern approach needs to be tested to see whether they are indeed more effective than guidelines.

[5] http://www.cs.vu.nl/~martijn/patterns/index.html

8. Conclusions

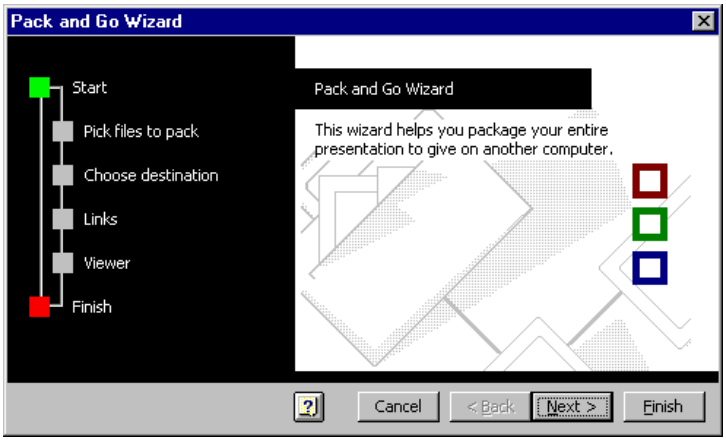
Patterns represent proven design knowledge in a much richer context than guidelines. Patterns are problem oriented and are potentially more usable for designers than guidelines. Patterns for UID require their own format and a standard template has been defined. The format is based on the other formats as used in architecture and Software Engineering but applied with a focus on designing for usability. We argued that only once a body of patterns has been accepted we could work towards a real pattern language for UI designers. The defined format and focus for UID patterns should contribute to the development of such a body of patterns.

References

1. Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I. and Angel, S.: A Pattern Language. Oxford University Press, New York (1977)
2. Bayle, E.: Putting it All Together: Towards a Pattern Language for Interaction Design. SIGCHI Bulletin. Vol. 30, No. 1 (1998) 17–24
3. Brown, W.J., Malveau, R.C., McCormick, H.W. and Mowbray, T.J.: Anti Patterns, Refactoring Software, Architectures and Projects in Crisis. John Wiley, New York (1998)
4. Dix, A., Abowd, G., Beale, R. and Finlay, J.: Human-Computer Interaction. Prentice Hall, Europe (1998)
5. Fitts, P.M.: The information capacity of the human motor system in controlling the amplitude of movement. Journal of Motor Behavior. Vol. 47 (1954) 381–391
6. Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading (1995)
7. Mahemoff, M.J. and Johnston, L.J.: Pattern Languages for Usability: An Investigation of Alternative Approaches. In Proc. of Asia-Pacific Conference on Human Computer Interaction APCHI'98 (Shonan Village, 1998). IEEE Computer Society, Los Alamitos, 25–31
8. Mahemoff, M.J. and Johnston, L.J.: Principles for a Usability-Oriented Pattern Language. In Proc. of Australian Computer Human Interaction Conference OZCHI'98 (Adelaide, 1998). IEEE Computer Societey, Los Alamitos, 132–139
9. Rijken, D.: The Timeless Way... the design of meaning. SIGCHI Bulletin. Vol. 6, No. 3 (1994) 70–79
10. Scapin, D.L. and Bastien, J.M.C.: Ergonomic criteria for evaluating the ergonomic quality of interactive systems. Behaviour & Information Technology. Vol 16, No. 4/5, (1997) 220–231
11. Smith, S. and Mosier, J.: Guidelines for Designing User Interface Software. MITRE (1986)
12. Sutcliffe, A. and Dimitrova, M.: Patterns, Claims and Multimedia. In Proc. of Conf. on Human-Computer-Interaction Interact '99 (Edinburgh, 30th August - 3rd September 1999). IOS Press. 329–335
13. van Welie, M., van der Veer, G.C., and Eliëns, A. Breaking down Usability. In Proc. of Conf. on Human-Computer-Interaction Interact'99 (Edinburgh, 30th August - 3rd September 1999). IOS Press, 613–620

Appendix

Name	**The Wizard**
Problem	**The user wants to achieve a single goal but several decisions need to be made before the goal can be achieved completely, which may not be known to the user.**
Usability Principle	User Guidance
Context	The Wizard pattern can be used when a non-expert user needs to perform an infrequent complex task consisting of several subtasks in a linear order where decisions need to be made in each subtask. The number of subtasks must be small, e.g., typically between ~3 and ~10.
Forces	• The user needs to perform a complex task but may not be familiar with the steps that need to be performed. • Each task needs to be performed but the users may not always be interested in each task. • The time it takes to perform the entire task. • The task are ordered but are not always independent of each other i.e. a certain task may need to be finished before the next task can be done.
Solutions	**Take the user through the entire task one step at the time. Let the user step through the tasks and show which steps exist and which have been completed.** When the complex task is started, the user is informed about the goal that will be achieved and the fact that several decisions are needed. The user can go to the next task by using a navigation widget (for example a button). If the user cannot start the next task before completing the current one, feedback is provided indicating the user cannot proceed before completion (for example by disabling a navigation widget). The user should also be able to revise a decision by navigating back to a previous task. The user is given feedback about the purpose of each task and the user can see at all times where (s)he is in the sequence and which steps are part of the sequence. When the complex task is completed, feedback is provided to shown the user that the tasks have been completed and optionally results have been processed. Users that know the default options can immediately use a shortcut that allows all the steps to be done in one action. At any point in the sequence it is possible to abort the task by choosing the visible exit.

Examples	 The user wants to package a presentation so that the presentation can be given on another computer. Several relevant decisions need to be taken and the wizard helps the user take these decisions. The green box shows the current position in the sequence of tasks.
Usability Impact	Improves the learnability and memorability of the task but may have a negative effect of the performance time of the task. When users are forced to follow the order of tasks, users are less likely to forget important things and will hence make fewer errors.
Rationale	The navigation buttons show the users that they are navigating a one-dimensional space. Each task is presented in a consistent fashion enforcing the idea that several steps are taken. A simple task sequence informs the user at once which steps will need to be taken and where the user currently is.
Known Uses	Microsoft Powerpoint, Pack and Go wizard; Installshield installation programs