# A Categorization of HCI Patterns

*Author*: David Carlsson
*Supervisor*: Jürgen Börstler
*Date*: 2004-08-17

Department of Computing Science
Umeå University
Sweden

## Abstract

Patterns are a way to package and recycle knowledge and experience instead of actual code. Patterns in software development were made popular by the seminal work of Erich Gamma and colleagues who draw on the work of the building architect Christopher Alexander. Since then the patterns movement has spread to many other software development areas. Today we have patterns for specific life-cycle phases (e.g. analysis patterns), development or quality issues (e.g. security, concurrency), application domains (e.g. web applications), and the teaching of computer science (pedagogical patterns).

The basic idea of the pattern movement in computing science is to spread the fact that patterns are a way to recycle knowledge and experience instead of code and modules. It is an approach to develop and reuse frameworks. Patterns are not invented, they are discovered in the real world and documented in an easily accessible and understandable form, so others can take part of the information.

In this thesis, we deal with patterns for Human Computer Interaction (HCI). HCI patterns are a tool for documenting knowledge of interface design. They will change the way we design interfaces in applications. HCI patterns provide designers with a common language and more professional documentation to work with than traditional guidelines do..

The main contribution of this thesis is a categorization of HCI patterns. The proposed categorization makes it possible to describe HCI patterns in a systematic way. When the number of patterns in HCI is growing large a sorting principle will be urgent. A categorization is also a necessity for an application where pattern developers could contribute with new patterns and interface developers could find and use already existing ones. We focus on two views when defining the categorization in this thesis; the pattern developer's view and the pattern user's view. To show the applicability of our approach we classify 73 HCI patterns from the literature using our classification scheme.

# Contents

# 1 Introduction

When the concept of object orientation was new, many believed in recycling of code or modules. The following quotes, which examplify the confident researchers and system developers felt for the technique, are taken from Rob Mattison's book *The Object Oriented Enterprise*:

> "object orientation is the first methodology with an universal applicability", "a universal template that can be applied to all aspects of computer systems development" and "since the principles can be applied to all phases and aspects of a system, they can serve as the one true glue that can hold our systems together". [MAT94]

A few years later many problems occurred within this area. Databases with recycled code were hard to maintain, the new projects were often very different from the former ones and there took a lot of time and knowledge to develop recyclable modules. These factors and the work of the architect Christopher Alexander led us into a new area called design patterns.

Design patterns are a way to recycle knowledge and experience instead of code and modules, it's a method to develop and reuse frameworks. This concept found its way into the world of object orientation where it's now considered as common knowledge. Now, this concept is on its way to change the way we design interfaces in applications. In other words, patterns are now used in Human Computer Interaction (HCI) to document knowledge of interface design. When design patterns begun to be used in the development of object oriented systems many patterns were written in a short period of time, and a need for a sorting principle became urgent. The same thing is now happening in HCI which means that there has to exist some way of sorting them out. This could be done by defining a categorization combined with an application that can help designers to search among the available patterns. As the number of HCI patterns increases, the need for sorting them arises.

The main goal of this thesis is to develop categories for HCI patterns. These categories will be defined in a way that existing and new HCI patterns can be organized in a structured way using an application. In this application pattern developers could contribute with new patterns and interface developers could find and use already existing HCI patterns when designing the interface of an application.

To demonstrate that the categorization could be applied to existing patterns, a number of patterns will be classified and described so they could be used in such an application. A background on patterns in general and a motivation of why HCI patterns will make life easier for both professional and amateur designers are also subjects of this thesis.

This thesis is organized as follows. In section 2 the patterns and pattern languages are described. Section 3 goes into depth in the area of patterns in HCI, which is followed in section 4 and 5 by a description of an application to support the management of patterns and the concept of pattern languages. In section 6 existing categorizations of HCI patterns and a new categorization is presented. This categorization will then be used in chapter 7 where the process of classifying 73 HCI patterns is described and exemplified. The thesis will then be concluded with a discussion in section 8 and a summary and conclusions in section 9.

# 2 Patterns and Pattern Languages

Design patterns began to find its way in building architecture in 1977 with the architect Christopher Alexander's book *A Pattern Languag*e [AIS77]. Gamma, Helm, Johnson and Vlissides (also called the *Gang of Four* or simply *Go*F) wrote the book *Design Patterns: Elements of Reusable Object-Oriented Softwar*e [GHJ95] 18 years later which was directed to the software community in computing science. Since then many books and papers covering this area have been published. The knowledge of patterns has now spread to many other areas than HCI, such as development of enterprise systems [YAN01], security of data systems [SPH02] and pedagogical aspects of teaching [PPP03].

## *2.1 Patterns in Architecture*

The first pattern language was invented and described by the architect Christopher Alexander, in the book *A Pattern Language* [AIS77]. The patterns in that book are sorted in a pattern language. This means that the patterns are arranged in hierarchies with the largest scaled pattern ranked highest, i.e. aligned at the top of the hierarchy, and the smallest scaled pattern is aligned at the bottom of the hierarchy. When using a pattern language, the reader starts with a high-scaled pattern and with the help of that pattern chooses a pattern of lower scale. This process is repeated until the user is satisfied with the information collected.
In *A Pattern Approach to Interaction Design*, Jan Borchers describes the content in Alexander's book in a very precise way:

> "It presents 253 design patterns of "user-friendly" solutions to recurring problem in urban architecture. They range from large-scale issues (COMMUNITY OF 7000, IDENTIFIABLE NEIGHBOURHOOD), via smaller-scale patterns (PROMENADE, STREET CAFE) down to patterns for the design of single buildings (CASCADE OF ROOFS, INTIMACY GRADIENT, SITTING WALL)." [BOR01, page 13]

The patterns in *A Pattern Language* are written down in a very structured way, every pattern follows a model to keep the conformity among the patterns. That means that every pattern consists of the same parts, presented in the same order:

1. **Number**. The number in order of the pattern, ranging from 1 to 253.
2. **Name**. The name of the pattern is very important. It should be a unique name that captures information about the pattern in a few words. The name should be very easy to remember.
3. **Ranking**. A ranking from one to three indicating how confident the author is in the pattern.
4. **Picture**. The picture shows a representation of a situation when the pattern is applied.
5. **Context**. The context describes in which environment the pattern should be applied. This means that it contains links to patterns of larger scale, which defines the context this pattern should be used in on a higher level of abstraction.
6. **Problem Statement**. The problem statement describes the situation that the pattern addresses.
7. **Problem Description**. The problem description is a thorough description of the problem that motivates the existence of the pattern.
8. **Solution**. The solution is a clean concise instructive solution to the problem.
9. **Diagram**. The diagram is a visualization of the solution that catches the main idea in a simple diagram.

10. **References**. The references contain references to smaller-scaled patterns in the pattern language.

The idea behind these patterns was to capture the knowledge of designing buildings and write it down in a structured form so it can be read and used by laymen. Alexander's goal was to help people design and build buildings so that the buildings inherited *The Quality Without a Name* or shorter *qwa*n. The qwan for Alexander was a combination of all the good attributes that a building could have, it was a way to express the feeling when the building felt perfect. To put words on that feeling, Alexander came up with the attribute qwan, a feeling of perfection that couldn't be described in any other words. In the foreword of Richard Gabriel's book *Patterns of Softwar*e, Alexander puts his goal of patterns in a very short and simple way:

> "In architecture, the question, the question I have been asking is very simple: "Can we do better? Does all this talk help to make better buildings?"" [GAB96, page vi]

This is the kind of question that should be prompted when talking about patterns in every area, not only in architecture.

### *2.2 Patterns in System Design*

Christopher Alexander describes the design patterns in his work as follows:

> "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, without doing it the same way twice." [AIS77, page x]

This description of patterns in architecture could very well have been a description of patterns in system design. The similarity to the way Gamma, Helm, Johnson and Vlissides (also called the *Gang of Four* or simply *Go*F) describes the patterns in their book *Design Patterns: Elements of Reusable Object-Oriented Software* [GHJ95] is very clear, although they describe patterns in system design:

> "The design patterns in this book are descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context." [GHJ95, page 3]

Patterns are now widely used in computer science, specifically in the design of systems. The book that caught the interest of the system design community was the above mentioned book by GoF. Since the release of this book the research on patterns in computer science increased. The patterns in this book are not sorted in a pattern language; they are gathered in a collection where the patterns are classified in three categories:

- Creational Patterns
- Structural Patterns
- Behavioral Patterns

An example of a pattern in this area is the *SINGLETON* pattern, which describes how to design a class that is only allowed to be instantiated exactly once. The *SINGLETON* pattern

belongs to the creational patterns along with five other patterns. Altogether the GoF presents 23 patterns which are still referenced in literature as the foundation of patterns in system design. The form of the patterns are different from the patterns used in architecture, GoF used the following scheme to describe their patterns:

1. Name
2. Intent
3. Also Known As
4. Motivation
5. Applicability
6. Structure
7. Participants
8. Collaborations
9. Consequences
10. Implementation
11. Sample Code
12. Known Uses
13. Related Patterns

Recently a number of books were published that combines the Java programming language with the use of design patterns [COO00, GRA99]. For more information about patterns and to find references to other books published in this area, see the pattern communities' homepage [PHP03].

### *2.3 Patterns in Other Areas*

After the Gang of Four presented their collection of patterns, several other areas have adopted the concept of patterns. This section will present a number of these different areas, to give an understanding of how patterns can be applied in other ways.

GoF presented 23 patterns to design object oriented applications in general. These can be used as a foundation to document patterns in specific areas. Bin Yang has defined a pattern language for developing J2EE applications that he calls E++ [YAN01]. J2EE stands for Java2 Enterprise Edition and it's a platform for developing enterprise applications in the Java programming language. His words will make justice for the work he has done:

> "E++ makes design patterns friendlier to designers by effectively coordinating patterns in real projects. Different from the GoF's taxonomic approach, the E++ pattern language contains not only design patterns, but also architectural patterns that provide frameworks of the whole application for design patterns. Moreover, it describes the rules for design patterns working together to construct a high-quality J2EE framework. In short, E++ shows how to architect a real world J2EE project using closely related design patterns." [YAN01]

E++ is not the only attempt to create patterns that cover the area of J2EE applications; a lot of other patterns in this area are collected at *TheServerSide.com* [JDP03]. Another area where patterns could contribute to a potential improvement is teaching and learning. A project called *The Pedagogical Patterns Project* where the goal is to develop a pattern language has been started. The aim for pedagogical patterns is to capture the expert knowledge of teaching and

learning. These patterns are supposed to simplify the process of transferring knowledge from senior instructors to juniors [PPP03]:

> "In essence a pattern solves a problem. This problem should be one that recurs in different contexts. In teaching we have many problems such as motivating students, choosing and sequencing materials, evaluating students, and the like." [PPP03]

Today the project has developed around 20 patterns, which are posted at the project web site. Joseph Bergin developed 14 of these patterns and arranged 10 of these in a pattern language [BER01]. Every pattern is composed in a similar way, following a scheme. Some parts in the scheme are similar to the schemes of patterns in other areas, but others are specific to pedagogical patterns:

1. **Name**. The name of the pattern.
2. **Thumbnail**. A short overview of the problem that the pattern addresses.
3. **Audience/Context**. The context where the pattern should be applied and what sort of audience the pattern addresses.
4. **Forces**. The motivation to why the pattern could be applied in the given context.
5. **Solution**. The solution to the problem at hand.
6. **Discussion/Consequences/Implementation**. Discussions regarding the solution, consequences of the solution and guidelines to how the solution should be applied.
7. **Special Resources**. Resources that the user of the pattern must allocate to use the pattern.
8. **Related Patterns**. Connections to other, related patterns in the pattern language.
9. **Example Instances**. Examples of situations where the pattern could be applied.
10. **Contraindications**. Information about when the pattern should not be applied.
11. **Acknowledgements**. Acknowledgement to people whom has contributed to the pattern.
12. **References**. References to material used in the pattern.

Patterns are widespread today and will probably expand more. A few examples of other areas where patterns are used follows:

- **Organizational Patterns**. Describes the structure and practices of human organizations.
- **Business Patterns**. Describes the key business purpose of a solution.
- **Programming Language Specific Patterns**. Describes design and implementation issues directed to a specific programming language.
- **Security Patterns**. Describes a particular recurring security problem that arises in specific contexts and presents a well-proven generic scheme for its solution [SPH02].

This thesis will focus on patterns from one area that is not mentioned above, HCI Patterns, which will be exploited more thoroughly in the next section.

# 3 Patterns in Human Computer Interaction

Design patterns in HCI were available a long time ago, but there hasn't been any depth going research until now. The first book that really focused on the area was *Patterns in Interaction Design* [BOR01], written by Jan Borchers. A couple of workshops have been held in the area before the release of that book, the content of those workshops are summarized in Borchers book. The focus in this section is to discuss why a good collection of HCI patterns is needed and how this can contribute and affect interface design. The definition of an HCI pattern I will use in this section and throughout the thesis is a definition that was agreed upon by the participants of CHI 2000 Workshop:

> "An HCI design pattern captures the essence of a successful solution to a recurring usability problem in interactive systems" [BOR01, page 179]

Today many companies use patterns as a complement in their software development [BCC96]. As mentioned before, patterns have made most progress in system design, or to be more precise object oriented design. Therefore these patterns are most widely used and HCI patterns have never been used in industrial development. In a few years time there will probably be a number of companies that use HCI patterns when developing applications. The problem today is that there is no well-accepted pattern language or collection of HCI patterns that developers can trust and use. The best attempt so far is Jennifer Tidwell's *Common Groun*d [TID99], but that collection is not accepted by all HCI designers. If a company would like to develop their own set of HCI patterns it would probably take to much time and effort, therefore the need for a good general pattern language or collection of HCI patterns would be welcomed

If such a collection existed some problems still remain and have to be discussed and solved before integration into the lifecycle of a software project will be possible. First and foremost agreements have to be made that the use of HCI patterns will improve some aspect of interface design. With the use of HCI patterns the number of hours spent on design might be far less than today, but the time saving is not a fact until HCI patterns are used in a number of real projects. Another possible implication is that the process of designing the user interface might become much easier than today, as the developer doesn't need a master's degree in cognitive science to do a good job. But, as stated earlier, serious effort has to be made to find out if this is the case or if it's only a utopia that interfaces will be developed faster and with a better result than today with the use of HCI patterns.

If patterns should be used in the process of interface design, the integration of patterns in the software lifecycle has to be discussed. Further, the concrete usage of this potential collection of patterns has to be thought about. A number of questions have to be stated; will the patterns only be there to be looked at when we have problems, or will it be used in the planning process? An important question is whether only HCI designer will be able to use the patterns, or if system developers in general get a chance to discuss the interfaces with arguments from the patterns? Even the end-users could with the help of HCI patterns criticize the graphical design of the system.

If the collection is growing real large, which a well accepted collection probably would, the need for some kind of search methodology will be needed. As will be discussed later, it's hard

to search among patterns; it's not easy to find the right keywords. Therefore, the right thing to do is to find some kind of categorization which the patterns in the collection can be classified according to.

Will only HCI people find the patterns that they are looking for or should developers in general, or even customers be able to find and use the patterns that they might have interest in? It has to be established that these different types of people also need different kind of search tools or different kind of categorizations. This is the main topic of this thesis and will be discussed in depth in later sections.

Another topic that has been discussed heavily around patterns and their usage is if pattern languages are the solution to how patterns are supposed to be used. Alexander presented his patterns in this way [AIS77], but the Gang of Four did not [GHJ95]. A pattern language is a collection of patterns where the patterns are linked to each other in a way that makes the language as a whole more useful than the patterns themselves. This is a very interesting topic and the implications of using a pattern language and how a language might be combined with an ordinary collection will be discussed further in this thesis.

### *3.1 The Need for a Common Language in HCI*

Humans are complex beings. It's not as straightforward as one might think to design a system which humans interact with. It's hard to create a good system without the engagement of end-users. When interaction designers are going to have a conversation with end-users it would be easy if everyone in the conversation understood each other without any problems. This is not the case today, users often have trouble expressing what they want and designers have trouble understanding the problems the users express. The problems don't get easier when customers also are involved in the process of designing a system.

If designers, users and customers all spoke the same language it would make the process a bit simpler, but it wouldn't solve all the problems that can occur during development of a system. A solution to the communication problem could be to use patterns as a common language entrance. All interaction patterns have a name and that name is the most important of the whole pattern. The name of the pattern should explain what the pattern does and be easy to remember and connect to the related problem. Those names could act as the pillars in a common language and the semantics of the pattern as a help to talk about more intrinsic details in the design. The good thing is that this new vocabulary would probably be of most use in a conversation between designers where it today don't exist a common language.

The problem, as mentioned earlier, is that it don't exist a widely accepted pattern language for interaction design or HCI today. As long as it don't exist such a common pattern language, any common spoken language between designers, users and customers would be hard to find and use. This problem will take a few years to solve but it could be done. A proof of that is the GoF patterns, which pattern names are used in several domains. Many system developers know what you talk about when you mention the *Abstract Factor*y. In system design and the object oriented world the main advantage of this sort of common language is when developers talk to each other. It's very optimistic to believe that it's possible to reach a customer or user when talking about objects and methods, and hopefully it's not important information for them either. So in that domain there is not necessary to create a common language that reaches the users of the system.

In interaction design people with different backgrounds are involved and this is why patterns have to be written in such a way that a user with little or no knowledge in HCI can understand the substance of the pattern. This was actually the first goal of Christopher Alexander, his patterns was meant to be developed in such a way that a layman could say how she liked her house to be built with the help of Alexander's patterns. This goal is not achieved in the patterns of GoF, but that was not the purpose with those patterns either, patterns of system design should not be understood by laymen, there is no point to make the patterns that way. As mentioned before, the users don't have to care how the system is built inside; they only need to care how the interface towards them is designed.

A pattern language is more important in interaction design than in pure system design. Unfortunately there is a long way there, but once there I think designing the user interface is going to be a lot easier thanks to a common language. Thomas Erickson wrote in the paper *Lingua Francas for Design: Sacred Places and Pattern Languages* about the need for a common language by the use of patterns in HCI. A fragment from his introduction shows his standpoint in the issue discussed in this section:

> "The idea is that a *lingua franca* is accessible to all stakeholders, particularly those who are traditionally marginalized in the design process: the users." [ERI00, page 2]

### *3.2 HCI Patterns vs. Guidelines*

A design pattern is a piece of text that explains a solution to a well-known problem and the purpose of a design pattern is to capture and recycle knowledge and experience. The same thing applies to guidelines, so why are we trying to make a pattern language in HCI when there already are many large collections of guidelines defined [MAY97, BRO88]?

The answer is that there are several differences between patterns and guidelines. Patterns are supposed to be structured; they consist of a number of fields that the developer of the pattern should fill in. Guidelines are very short and concise, which is a good thing if it's the fast solution to a concrete problem in a very well defined context that is demanded. The problem is to find the right guideline for both the context and the problem itself. Many guidelines are already documented with a context. But the hard thing is to find what the context of the guideline is; i.e. the context of the guideline is often not well defined. A pattern has a well-defined context, which makes it easier for a user to find what they are looking for. Ingjerd Skogseid and Michael Spring tried to create patterns from a number of guidelines. Their assumption was that there exist a lot more guidelines than there are patterns, which could imply that one pattern consists of a number of guidelines [SSP95]. The reason to why patterns are fewer is that a pattern often catches a larger problem than a guideline does. Constructing a pattern is not an easy task [VLI96] but Skogseid and Spring showed that it could be done with the help of guidelines so guidelines seem to have a connection to HCI patterns in that aspect.

The problem with guidelines being short and concise are that they often are hard to interpret, and again, thereby hard to find and select among. Both guidelines and patterns can be conflicting, but patterns are in conflict in another way than guidelines. Patterns have different solutions to the same problem. That doesn't mean that one of the solutions is better than the other, it only means that the user of the patterns can solve the problem in different ways by their own choice. Conflicting guidelines are of another character, they can really be opposite solutions of each other, i.e. if you follow one guideline, and you might break another.

The core meaning of a guideline can be hard to understand and interpret due to its compactness. A pattern has always numerous ways that a reader could attack and understand the core meaning of the pattern. The reader also gets a lot of different views of the problem that the pattern is supposed to solve, which of course helps the reader to understand the problem. Every human doesn't think and attack problems in the same way and therefore it's very good for readability to give different people multiple ways to understand a piece of text. If it's hard to understand a guideline right away, different contexts where this problem might have been seen must be thought about. Patterns make this job for you; the context is already expressed explicitly in the pattern.

A problem with all experienced-based tips is the factor of trust. A pattern has a rationale section where the author of the pattern explains why this pattern works in terms of laws of nature so that the reader knows that the solution to the problem is well founded. A guideline does not often include a section where the rationale of the guideline is presented, which means that the user has to find out the rationale him self.

Patterns make facts explicit around problems and solutions. Guidelines are abstract claims that force the user to create most of the things that is missing between a guideline and a pattern themselves. This must be done if the user of the guideline wants to be sure that the guideline is correct and that the context of the guideline is well understood.

It's probably not possible to translate every guideline into a pattern, and all users don't do that every time they use a guideline. If a user wants to be sure that she found the right guideline she has to think about a lot of things herself. The author of the guideline knew all these things when she documented the guideline, so why not put this information there explicitly. It would make the interpretation of the guideline so much easier. If the guideline writer documented all these things when she created the guideline it would still not for certain be a pattern, because that doesn't make the solution a proven one. But it helps to make the point that the solution might not be proven so that the user could turn to another guideline or a pattern to get a proven solution to the problem. Martijn van Welie, van der Veer and Eliëns summarize the discussion of pattern vs. guidelines in their paper *Patterns as Tools for User Interface Design*:

> "Patterns explicitly focus on context and tell the designer *whe*n, *how* and *why* the solution can be applied. Hence, patterns can be more powerful than guidelines as tools for designers." [WVE00, page 1]

### *3.3 An Application to Handle Patterns*

An application to handle patterns is a necessary condition for the process of using patterns in real projects and creating and maintaining a large collection of patterns. Such an application could be used for a number of actions as writing, editing and commenting patterns. The interest for this discussion lies in both writing and using patterns with the help of an application. When a user uses patterns in a development process, the patterns have to be classified so the user can find the patterns of interest. This means that the author of the patterns has to specify which categories the pattern should belong to.

### 3.3.1 Pattern Editing Tool

Borchers has in his book *A Pattern Approach to Interaction Design* [BOR01] discussed an application to handle patterns which he calls Pattern Editing Tool (PET). This application is not yet implemented; but he makes a couple of good points to what a tool for editing and using patterns should be able to do. In the end of July 2001, researchers from different areas of interest started a project of building PET. So far nothing except a couple of use cases is made [IPJ01], but it will probably happen a lot in the future.

# 4 A Categorization of HCI Patterns

As patterns will be used more in the future, they have to be easy to find and use. An application to handle patterns is necessary for an integration of patterns in the lifecycle of software development. To create an application that offers more than a simple unorganized listing of existing patterns, a categorization that separates patterns from different domains (i.e. system design domain and interaction design domain) from each other is a necessity. In the article *Zur Klassifikation von Patterns* the authors Peter Fettke and Peter Loos discuss a domain categorization of patterns [FLO01]. Further it's assumed that a categorization dealing with the domain level already exists, therefore the rest of the thesis deals with a HCI pattern categorization. Several reasons for constructing a categorization have already been sorted out. These reasons are based on the assumption that therein the future will exist several hundred HCI patterns:

- **The pattern developer's reason**. A pattern developer thinking of documenting a pattern might like to know if this pattern is documented before. Therefore a categorization where he can classify his own pattern and look in the same category to see if a similar pattern already exists is needed.
- **The HCI expert's reason**. An HCI expert developing interfaces might like to extend the affordance of some part. This could by looking in several patterns to get tips on how to do this in a good and well-documented way. Without a categorization that maps terms like affordance it would be hard to find the patterns of interest.
- **The interface developer's reason**. An interface developer developing some distinct part of an interface would be helped by reading an HCI pattern that has documented problems regarding that part. Therefore a categorization where patterns are classified by interface parts would suite the interface developer.
- **The interface user's reason**. An interface user could use patterns to understand the reasons why designers and interface developers chose the design they chose. Another reason for the user to use patterns is to be able to discuss design issues with the interface developers. Interface users will probably search among patterns by parts in the interface, or so-called widgets.

These reasons imply that one sort of categorization will not be enough; at least two different types of categories with multiple attributes must exist.

Several papers touch the subject of categorizing HCI patterns. In the next section pattern languages will be discussed, which is a sort of categorization. After that existing categorizations that I find important will be discussed, followed by a description and discussion of my own categorization and why I chose the attributes that I chose.

## *4.1 Pattern Languages*

As discussed in the section *history of patterns*, patterns were in the beginning introduced with help of the concept pattern languages. In a pattern language, patterns are grouped as a collection with references between patterns. Patterns always reference other patterns on different levels of abstraction; patterns on the same level don't reference each other explicitly.

Pattern languages are an approach to use when constructing artifacts; they approach the problem in a natural way. The designer starts on a level where the patterns don't include any details, and follows the references from the chosen pattern chosen down to a lower level of abstraction. This is the way people think when constructing artifacts. To start with planning where a chair in a street café should be located without planning how the city should look like first, would be a hard way of doing architecture.

Pattern languages are not enough when using patterns in a development environment. If the only way to use patterns were through pattern languages, the whole process of constructing an artifact is dependent on patterns. Patterns should be a help to the already existing development techniques, not a replacement. Therefore there should be an easy way of finding patterns to solve the given problem.

A solution could be to classify patterns in order to make it possible for users to look in the categories matching the problem and find a pattern dealing with the problem. Next section will deal with such categories among HCI patterns.

## *4.2 Existing Categorizations*

The existing categorizations deal with the problem from different views. This section will describe the existing categorizations that are relevant for the categorization of HCI patterns. Different authors' approach to HCI patterns will also be accounted for when this is relevant for the categorization in question.

### 4.2.1 Martijn van Welie and Hallvard Traetteberg

In the paper *Interaction Patterns in User Interfaces* van Welie and Traetteberg [WTR00] discuss and present several interaction patterns. These patterns are created with the approach that when creating a collection of HCI patterns, the pattern developer should keep in mind that there is a distinction between users and designers perspective of how an HCI pattern should be used. They also state that an HCI pattern should be in a format that focuses on usability, i.e. the patterns should take the users perspective. This implies that it becomes important how and why the usability is improved. Van Welie and Traetteberg mean that by focusing on usability it becomes easier to see whether or why the solution is a good and possibly accepted solution. This quote from the paper summarizes their opinion of users and designers perspective of problems in user interface design (UID):

> "Certain solutions in UID solve problems that designers or marketing people have but *not* necessarily problems users have. E.g. banners and splash screens are accepted designs, but hardly for usability reasons. It's not difficult to identify patterns in user interfaces but it's hard to identify those patterns that *really* benefit the user, and explain the usability aspect." [WTR00, page 1]

Their pattern collection is classified according to the kind of usage problem the patterns address. The user interface principles they, are taken from Norman's *The Psychology of Everyday Things* [NOR88]. These principles touch the kind of problems and questions that the user might have when interacting with a system. Below are the principles together with explanations:

- **Visibility**. Gives the user the ability to figure out how to use something just by looking at it.
- **Affordance**. Involves the perceived and actual properties of an object that suggest how the object is to be used.
- **Natural mapping**. Creates a clear relationship between what the user wants to do and the mechanism for doing it.
- **Constraints**. Reduces the number of ways to perform a task and the amount of knowledge necessary to perform a task, making it easier to figure out.
- **Conceptual models**. A good conceptual model is one in which the user's understanding of how something works corresponds to the way it actually works. This way the user can confidently predict the effects of his actions.
- **Feedback**. Indicates to the user that a task is being done and that the task is being done correctly.

The principles listed above assume that the user acts rationally at all times. Humans are not always rational, which means that there must exist some additional principles that deals with these user mistakes:

- **Safety**. The user needs to be protected against unintended actions or mistakes.
- **Flexibility**. Users may change their mind and each user may do thing differently.

Van Welie and Traetteberg focus on usability in their patterns. Because of this they included usage indicators in their *rationale* section. Every pattern has a section with an explanation to why and how the pattern approves the usability. They have not used the usage indicators as a foundation for their categorization of the collection, but this way of looking at HCI patterns is unique. The usability indicators are lined up below together with comments from the authors:

- **Performance Speed**. How fast users can work with the system.
- **Learnability**. How easy the system is to learn.
- **Memorability**. How well users remember how to use the system.
- **Satisfaction**. The satisfaction users get when using the system.
- **Task Completion**. How much of the task could be completed.
- **Errors**. The number of error users made.

The authors' point of view is that a pattern should make impact on at least one of these usage indicators in order to be called an HCI pattern.

### 4.2.2 Michael J. Mahemoff and Lorraine J. Johnston

Mahemoff and Johnston discuss four kinds of patterns in HCI [MJO98], there partitioning of patterns can be seen as a categorization. The four categories that the authors describe are listed below together with the explanations to each category:

- **Tasks**. This type of patterns is focused on the tasks that users will perform in interaction with the system.
- **Entire Systems**. Patterns of entire systems capture the issues of the whole system. This type of patterns lies on a very high level of abstraction.

- **User Interface Elements**. The purpose of these patterns is to describe the area of usage for user interface elements, popularly called ”widgets”. This category is on a very low level of abstraction.
- **Users**. Patterns of users deal with the profiling of users. They can be seen as a way to explore the forces involved in the context of a particular kind of user interacting with the system.

These types of patterns, or categories, are on a very high level. The categories that touch HCI patterns are the types that deal with entire systems and user interface elements. This is a good start of a categorization, the categories should though be divided up further, as the authors imply in the paper.

### 4.2.3 ChiliPLoP’99

Another categorization was created as a result of a workshop at ChiliPLoP’99, the first workshop where the pattern communities from software engineering and HCI got together [BOR01]. The categorization is built on three main dimensions, where the most important dimension is *level of abstractio*n, which is divided in three parts, or attributes:

- **Task**. The highest level of abstraction. Patterns on this level deal with a complete task of the user.
- **Style**. The middle level of abstraction. On this level patterns that deal with the style of a certain part of the interaction are placed.
- **Object**. The most concrete level of abstraction. The most concrete patterns that deal with low-level questions of user interface design are placed on this level.

The second dimension in this taxonomy is *functio*n that deals with the classification of patterns with respect to what functions the pattern addresses. The different functions, i.e. the attributes, that this dimension consists of are:

- **Perception**. Here are patterns that address questions about input and output from the interface classified.
- **Manipulation**. Here are patterns that deal with manipulation of some kind of application data classified.
- **Navigation**. Here are patterns that deal with the users navigation in the system classified.

The third dimension is *physical dimensio*n. This dimension deals with how patterns address questions of time and space. The three attributes that this dimension consists of are:

- **Space**. Here are patterns that deal with spatial layout classified.
- **Sequential**. Here are patterns that deal with issues where the actions occur in sequential order classified
- **Time**. Here are patterns that deal with time in a continuous way classified.

The ChiliPLoP’99 categorization was the first to define categories in different dimensions.

### 4.2.4 INTERACT'99

At the INTERACT'99 conference on HCI there was another workshop which dealt with patterns in HCI [BOR01]. Several organizing principles for HCI design patterns were conceived and elaborated in detail, two of these will be presented here.

The first principle, which is called the scale principle, uses *scale* as a way to classify patterns. A number of patterns were submitted to the workshop as a basis for the discussions. These patterns were successfully sorted according to the principle of scale. These were the resulting categories:

- Society (beyond systems)
- Multiple Users
- Social Position
- System
- Application
- UI Structure (Dialogue)
- Components (containers, windows, layout)
- Primitives (buttons and other simple widgets)
- Physical properties

The other organizing principle that was a result of the workshop is developed to fit in a HCI design process and is called the *process principle*. The categories in this principle are:

- Analysis oriented levels
- Culture and Society
- Use and Navigation
- Structural levels

## *4.3 The Need for Several Categorizations*

The main reason for a categorization is to help people find patterns in a database, but problems arise when different groups of users use the database. To create one categorization with the aim to meet all demands of pattern developers, HCI experts, interface developers and interface users is very difficult. The main reason for that is the users searching for patterns are on different levels of knowledge. They also have different reasons for searching for patterns.

The pattern developers need a categorization that help them find existing patterns so they can see if the patterns are already written. A pattern developer could also be interested in existing patterns to get some ideas how to write and formulate his own.

HCI experts need a categorization that helps them find patterns that increase usability by some means in the interface. An HCI expert does often do tests on a potential user group to see if the interface meets the demands specified. If some part of the interface fails to meet these demands, there is a need to look for a pattern that helps the expert to increase this aspect of the interface.

A categorization that suites the HCI expert could be the same categorization that the pattern developer need. Both groups search for patterns by advanced HCI terms based on usability aspects. These two roles are often held by the same person. After all, pattern development is a process where an author documents his experience in a structured way, so that other people can take advantage of it.

Both an HCI expert and an ordinary developer that is developing interfaces could use HCI patterns to get some ideas how to develop an artifact that is easy to use. Both groups could also use HCI patterns to reach a large repository of experience that is documented in a structured form. To find these patterns, there should exist a categorization that sorts the patterns by the parts in the interface that they have solutions to, not on advanced HCI concepts. Therefore this categorization is not the one that the pattern developer or the HCI expert needs to use when defining how the pattern increases the usability of some part of the interface. Such a categorization wills on the other hand suite the interface user that also targets on how to design a distinct part of the interface. This categorization, that the interface developer and interface user would like to use, is a different categorization and therefore there must exist two different categorizations.

The first categorization will be called *The Pattern Developers Categorization* and the other *The Pattern Users Categorizatio*n.

## *4.4 The Pattern Developers Categorization*

As discussed earlier pattern developers and HCI experts need a categorization of HCI patterns. One reason for this is that developers would benefit by knowing if the pattern they are creating is documented or not. Another reason is that a categorization could help developers to see that the pattern really improves the usability of the system. An HCI expert's reason for using a categorization is similar to the reasons pattern developers have. The expert needs to find patterns that improve the usability of the interface in some aspect.

*The Pattern Developers Categorization* is divided into two parts, where both parts consist of several attributes. The first part is called *usability* and second is called *effectivity*. *Usability* is focused on usability and consists of five attributes and *effectivity* is focused on measurable human factors and consists of five attributes.

### 4.4.1 Usability

The five attributes of *usability* are all refinements of the more abstract attribute, with the same name as this part, *usability*. When pattern developers classify patterns using *usability* they ask themselves the question: In what way does this pattern improve the usability of the interface? The answer to this question should be at least one of these attributes; otherwise I find it hard to believe that the pattern in question is a HCI pattern. These are the attributes of *usability*:

- **Visibility**. The pattern uses principles that give the user a feeling that objects in the interface appear clearer.
- **Constraints**. The pattern uses principles that use delimitations or constraints that give the user a better feeling of the interface.
- **Affordance**. The pattern uses principles that take advantage of the "force" which every artifact has to give the user a better feeling of the interface.

- **Mappings**. The pattern uses principles that make the user catch the connection between a thing in the interface and an artifact in the "real world". This should lead to the user getting a better feeling for the interface.
- **Feedback**. The pattern uses principles that give the user a sign that something has happened in the system.

Usability is a key concept in HCI. The way to improve usability is to make a system easy to learn and easy to use [PRE94]. In order to develop systems with good usability, the designers strive to use the primary design principles that according to Donald Norman are visibility, affordances, constraints, mappings and feedback [NOR88].

### 4.4.2 Effectivity

When pattern developers classify patterns according to *effectivity* they ask themselves: Which human factors increase when someone uses the pattern to change or develop an interface? The answer should be one or more of the attributes in *effectivity*:

- **Time to Learn**. How long does it take for typical members of the user community to learn how to use the commands relevant to a set of tasks?
- **Speed of Performance**. How long does it take to carry out the benchmark tasks?
- **Rate of Errors by Users**. How many and what kind of errors do people make in carrying out the benchmark tasks? Although time to make and correct errors might be incorporated into the speed of performance, error handling is such a critical component of system usage that it deserves extensive study.
- **Retention over Time**. How well do users maintain their knowledge after an hour, a day, or a week? Retention may be linked closely to time to learn, and frequency of use plays an important role.
- **Subjective Satisfaction**. How much did users like using various aspects of the system? The answer can be ascertained by interview or by written surveys that include satisfaction scales and space for free-form comments.

The five attributes of *effectivity* are taken from Ben Shneiderman's book *Designing the User Interface* [SHN98], where he defines five measurable human factors that are central to evaluation of an interface.

## *4.5 The Pattern Users Categorization*

There are several needs for a categorization that is focused on what the user of a pattern collection demands. The user could not be considered an HCI expert, which makes it hard to use categories like *perceptual affordance* and *semantic constraint*s. Users like that need a categorization focused on end-users, contexts and tasks. Further they need a categorization that help them find patterns based on what part of the interface they are going to develop.

*The Pattern Users Categorization* is divided into three parts, where all parts consist of several attributes. The first part is called *experience*, the second *interaction style* and third *interface level*. Experience is focused on what sort of end-users the target systems have and it consists of three attributes. *Interaction style* is focused on the interaction style of the target systems and consists of five attributes. *Interface level* is focused on the interface levels of the target systems and consists of three attributes.

### 4.5.1 Experience

When pattern developers classify patterns according to *experience* they should ask themselves: Which user group or user groups is this pattern attended for? The answer should be one or more of the attributes in *experience*:

- Novice or first-time Users
- Knowledgeable Intermittent Users
- Expert Frequent Users

The three attributes of *experience* are taken from Ben Shneiderman's book *Designing the User Interface* [SHN98].

### 4.5.2 Interaction style

When pattern developers classify patterns according to *interaction style* they ask themselves: Does this pattern improve, simplify or make it any easier for the user to supply or retrieve data to or from the target system? If so, which type of interaction style does the pattern improve? The answer should be one or more of the attributes in *interaction style*:

- Direct Manipulation
- Menu Selection
- Form Fillin
- Command Language
- Natural Language

The five attributes of *interaction style* are taken from Ben Shneiderman's book *Designing the User Interface* [SHN98].

### 4.5.3 Interface level

When pattern developers classify patterns according to *interface level* they ask themselves: Which level of the interface does this pattern affect? The answer should be one or more of the attributes in *interface level*:

- Application
- Containers
- Widgets

# 5 The Classification of 73 HCI patterns

In this section 73 patterns will be classified according to *The Pattern Developers Categorization* and *The Pattern Users Categorizatio*n. The difficulty lies in showing that the categorizations work well when different authors are involved in the patterns. Therefore one pattern language and one collection of patterns will be classified. Van Welie and Traetteberg created the collection [WTR00] and Tidwell constructed the pattern language [TID99].

The forms of these patterns are different. A new compact form that fits patterns from the pattern language and the collection is created. This form focus on the categorizations and how it's possible to compare patterns that originally are in different forms.

A goal with this form is that every pattern described should at least be understandable for a person that has seen the situation before. Most of the situations are easy to recognize when reading the pattern. If there is a problem of recognition a reference to the pattern in its original form is supplied. The original form that the authors of these patterns are using is much clearer than the compact form, but as these patterns are accessible from the source no information will be lost.

For every pattern, the problem and answer to the questions *whe*n, *how* and *why* the solution can be applied to that problem will be stated. The goal is to take as much information from the originals as possible, but sometimes sentences have to be reformulated. The reformulations are not an attempt to improve the patterns in any way; it's needed to make the patterns fit in the compact form.

The original's *problem* section will without exceptions be used as the *problem* section in the compact form. Both authors' *problem* section is often very short and there is no need to reformulate these.

The *when* section will always be van Welie's and Tidwell's *context* section, the only difference is that the compact form always starts with the word *Whe*n, which means that some sentences have to be rephrased to make them fit.

The *how* section is emphasized and is both authors *solution* section. The *solution* section of the original authors is often very detailed and the goal is to make the reader understand the concept of the pattern, not to encourage them to use the pattern in this compact form. Therefore only a small part of the originals *solution* section will be taken, which still delivers the point of the idea behind the pattern. The parts left behind are the small, but still important, details of the solution. Those details are necessary for the patterns and the readers are strongly recommended to read the patterns in its original form before usage.

The *how* section will always begin with the word *Then* followed by a comma, *then* is a word that fits better in this context than the word *therefor*e, which is often used when dealing with patterns.

The *why* section in the compact form is van Welie's and Tidwell's *forces* section. Both uses a list of items to represent their forces. They have approximately four or five points in their *forces* section. In the compact form one or two points will be used when representing the

whys. The following syntax is used: *A reason for this is that [why1]. Another reason is that [why2]*. The problem with this section is that van Welie has also a rationale section, which Tidwell has not. A rationale is according to the dictionary ”An explanation of the working of some device in terms of laws of nature” which is a good thing to know when dealing with patterns, sort of an explanation to why this pattern works. To make both authors patterns comparable their *forces* sections had to be used and van Welie’s *rationale* section had to be left behind. This results in a form that follows the following syntax:

*Problem:* [The problem that the pattern is supposed to solve]
*When, How and Why:* When [The context in which the pattern is to be used]. Then, [The solution to the problem in the given context]. A reason for this is that [An explanation to why the solution works in the given context]. Another reason is that [A second explanation why the solution works in the given context].

The compact form is created to show the classification and sometimes the choice of classification will be hard to understand without looking at the originals. The reference to where the original patterns is to be found is given next to the pattern name, it will not be any problem to find the pattern in its original form. Below is an example of a pattern (Form) in its original form as Jennifer Tidwell presented it in her collection. Following that is two patterns (Form and WYSIWYG editor) presented in the compact form described above:

**Form (original form) [TID99]**

*Examples:*
- Tax Forms
- Job application forms
- Ordering merchandise through a catalog
- Ordering a pizza over the phone

*Context:* The user has to provide preformatted information, usually short (non-narrative) answers to questions.

*Problem:* How should the artifact indicate what kind of information should be supplied, end the extent of it?

*Forces:*
- The user needs to know what kind of information to provide.
- It should be clear what the user is supposed to read, and what to fill in.
- The user needs to know what is required, and what is optional.
- Users almost never read directions.
- Users generally do not enjoy supplying information this way, and are satisfied by efficiency, clarity, and a lack of mistakes.

*Solution:* Provide appropriate “blanks” to be filled in, which clearly and correctly indicate what information should be provided. Visually indicate those editable blanks consistently, such as with subtle changes in background color, so that a user can see at a glance what needs to be filled in. Label them with clear short labels that use the terminology familiar to the user; place the labels as close to the blanks as is reasonable. Arrange them all in an order that makes sense semantically, rather than simply grouping things by visual appearance.

Visually, arrange the blanks and labels according to a spatial grid; align the edges of the stronger graphic elements, like boxes, where you can. Give them a neat look and a good visual rhythm, but don't let a strong geometry overwhelm the meaning of what is written or provided – big arrays of edit fields (or whatever) looks scary! If there are a lot of them, at least separate them into subgroups (Small Groups of related Things); again, do this according to the meaning of the information, not just what looks good.

Provide reasonable default values wherever possible, to lessen the amount of work that the user has to do (Good Defaults). If the user provides information that makes some parts of the form irrelevant, disable them (Disabled Irrelevant Things). Likewise, if the user provides some clue which enables a software-based form to predict what information will be filled in elsewhere, then have the software do it for them. For example, if a user is providing their address and phone number, the town they provide might only have one area code in it – so fill in their area code for them. But don't do it wrong; that's worse than not doing it at all, because the user then has to both notice the error and correct the form.

Be forgiving about the format of what the user provides, as much as is possible (Forgiving Text Entry). But if the required information simply must follow a certain format, then an empty, featureless text field is the worst possible thing to have to fill in. You never know if you're doing it right. Always offer a clue about what is expected by constraining the user's input in a reasonable way (see Structured Text Entry for an example), and don't resort to validating afterwards, giving the user a nasty message about how wrong their input was. Giving an example right there on the form is a better idea, but still not as good as visual constraint.

(Sometimes, cultural factors make visual constraints unnecessary. A login screen, for instance, usually just has fields for "User name" and "Password" – the possible input strings are constrained, of course, but 99,9% of the its users will understand exactly what is expected, and constraints or extra information will be more trouble than it's worth. See Don Norman's *The Design of Everyday Things* for lengthy discussions of physical and cultural constraints.)

*Resulting Context:* You must pick controls for each of the pieces of information to be supplied. These may include:

- Choice from a Small Set, if the user should pick one or more from a set of 10 or fewer choices (give or take).
- Choice from a Large Set, if there are more than 10 choices.
- Editable Collection, if the user should construct a list of items.
- Sliding Scale, if the possible values vary linearly between an upper and lower bound.
- Forgiving Text Entry, if the recipient of the information is capable of parsing whatever text the user provides, which could vary widely.
- Structured Text Entry, if the recipient needs the information in a very specific format and cannot parse a potentially wide vaiety of responses.

*Notes:* No one ever fills in forms for fun. They do it because they want something (as with a catalog order), or because they are compelled to do it (taxes), or because it's their job (data entry). It's pointless to be cute or clever, and if you make the user have to read extra instructions, or go back to redo something the misunderstood the first time around, or jump around the form to fill in things "out of order", they will be irritated.

If you can find a non-contrived way to use something other than a form, do so. For instance, why ask someone to laboriously fill in a day, month, and year when you can just have them pick a day from a calendar?

**Form (compact form) [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How should the artifact indicate what kind of information should be supplied, and the extent of it?
*When, How and Why:* When the user has to provide preformatted information, usually short (non-narrative) answers to questions. Then, provide appropriate ”blanks” to be filled in, which clearly and correctly indicate what information should be provided. Visually indicate those editable blanks consistently, such as with subtle changes in background color, so that the user can see at a glance what needs to be filled in. Label them with clear, short labels that use terminology familiar to the user; place the labels as close to the blanks as is reasonable. Arrange them all in an order that makes sense semantically, rather than simply grouping things by visual appearance. A reason for this is that it helps the user to see what is required and what is optional. Another reason is that it’s simple so there is not much need for directions, which the user seldom reads.

**WYSIWYG Editor (compact form) [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How can the artifact best present what is being created, and what the user needs to do to create or change it?
*When, How and Why:* When the artifact is used as a tool or environment in which other artifacts may be created (particularly those with visual aspects). Then, always show the user an accurate and up-to-date representation of the artifact they are creating (“what you see is what you get”); allow the user to interact directly with it as they add to it, delete from it, modify it, and so on. A reason for this is that people already have the skills to draw, write, sculpt, arrange, etc. Another reason is that the user wants to see what they’re creating immediately, as they work with it; this instant feedback leads to a strong sense of engagement with the work.

The remaining 71 patterns are located in appendix 1 in compact form.

All patterns are classified according to *The Pattern Developers Categorization* and *The Pattern Users Categorizatio*n. The classification of two patterns is presented below; classifications of the remaining patterns are to be found in appendix 2. I have chosen to use the sign X to mark that the attribute apply to that pattern and an O indicates that the attribute doesn’t apply.

**Table 1: The Pattern Developers Categorization, usability**

| Pattern | Visibility | Constraints | Affordance | Mappings | Feedback |
|---|---|---|---|---|---|
| Form | O | O | O | X | O |
| WYSIWYG Editor | O | O | O | X | X |

**Table 2: The Pattern Developers Categorization, effectivity**

| Pattern | Time to Learn | Speed of Performance | Rate of Errors by Users | Retention over Time | Subjective Satisfaction |
|---|---|---|---|---|---|
| Form | X | X | X | O | O |
| WYSIWYG Editor | X | O | X | X | X |

**Table 3: The Pattern Users Categorization, experience**

| Pattern | Novice or first-time Users | Knowledgeable Intermittent Users | Expert Frequent Users |
|---|---|---|---|
| Form | X | X | X |
| WYSIWYG Editor | X | X | O |

**Table 4: The Pattern Users Categorization, interaction style**

| Pattern | Direct Manipulation | Menu Selection | Form Fillin | Command Language | Natural Language |
|---|---|---|---|---|---|
| Form | O | O | X | O | O |
| WYSIWYG Editor | X | O | O | O | O |

**Table 5: The Pattern Users Categorization, interface level**

| Pattern | Application | Containers | Widgets |
|---|---|---|---|
| Form | O | O | O |
| WYSIWYG Editor | O | O | O |

# 6 Discussion

The hard part with creating a categorization of HCI patterns is where to start the work. After reading several articles and a number of books, that didn't only touch the subject of HCI patterns, I started my work by writing about patterns in general. When acquainting myself with patterns in architecture and patterns in object oriented programming; I realized how important it was to see a whole picture of patterns when working with a small part. Reading about patterns in object oriented programming gave me a view of how patterns successful in industry today look like. Studying patterns in general gave me knowledge and an understanding of how researchers for many years discussed how patterns could be presented, used and interpreted.

After taking part of and summarizing the information on patterns in several areas, the research on patterns in HCI started. The only published book that treats this subject is Jan Borchers' *A Pattern Approach to Interaction Design* that was released during the work of this thesis. As it goes through the history behind HCI Patterns and discusses its potential area of usage in detail it simplified my work drastically. Two articles that have affected the outline of this thesis are van Welie's *Patterns as Tools for User Interface Desig*n and *Interaction Patterns in User Interface*s where he discusses the usability perspective and several other aspects regarding HCI patterns.

The goal with the thesis was to create a categorization of HCI patterns and evaluate it by classifying a number of patterns to see if it was applicable. My knowledge in HCI combined with my experience in system development gave me a possibility to carry out the task at hand. The knowledge in system development simplified the process of understanding the usefulness and structure of GoF's patterns and thereby understand the need for patterns in HCI. I therefore had the possibility to see the categorization from both the users and the creators' perspective. This gave me the opportunity to use terms from recognized researchers in HCI to create a foundation for the categorization. The separation of *The Pattern Developers Categorization and The Pattern Users Categorization* has mainly been the same throughout the process. But the content in the two categorizations, i.e. the parts and the attributes, has changed many times during the classification of the patterns. An alternative of using X and O to mark if an attribute apply or not to a pattern is to use a graded scale, where the patterns are classified by how strong they apply to the attributes. This method could make the classification more exact but it would make the classification process much more difficult to perform.

The approach while creating this categorization was to start with a well-considered draft and begin to classify patterns in large scale. I was convinced that I would get new ideas and reject old ones while classifying the patterns. When mass-classifying patterns I realized it would be more conclusive if I used patterns from at least two authors. An obvious choice was Jennifer Tidwell's collection, which is the most recognized collection according to the literature. My second choice was also obvious because Martin van Welie's collection is the largest and most structured of the remaining. These collections were quite different in structure, which was good for the process of finding a categorization that suites HCI patterns written by different authors.

To be able to implement the classification of these quite different collections I realized the need to summarize the patterns on a compact form and in a syntax that makes it easier to compare patterns written by different authors. When constructing the compact form and writing some patterns following that syntax, I realized that this form is very suitable for an application that handles patterns. A requirement of such an application is that it consists of a database that is searchable where the patterns are stored. My categorizations and the compact form that I have used to present and classify the patterns are very suitable in this context. If the authors enter the patterns in the application using the compact form and classify them according to *The Pattern Developers Categorization* and *The Pattern Users Categorization* the patterns can be searchable for HCI experts, interface developers and users. Then we have a very good start to a usable application that could be used to handle HCI patterns in system development.

Further I believe that every creator of patterns should be free to write their patterns in any form they like. But it would be good for the usage of patterns if they could supply users with an classification and a compact form or ”teaser” that is common to all pattern authors. This would result in a more user-friendly system. When a user searches for patterns in the application he can choose the attributes suitable to his needs and get a list of patterns as answer. At first he can take a glance at the patterns in compact form and if he still is interested in using the pattern he can look at it in its original form. The original form doesn’t have to be similar for all patterns as earlier mentioned; it could be a document organized in any form that the author of the pattern likes. To force the creator to write his pattern in the same form and structure as every other pattern author is an impossible mission. Several patterns are already created in different forms and structures and different authors have different opinions on which the best form to present an experienced based tip is.

The application should, to be able to serve its purpose, contain some base functions as change, comment and delete patterns. There could even be more advanced functions as grouping of similar patterns and organizing patterns in a pattern language. My purpose with this discussion was to express the usefulness of having a common categorization of patterns and to have a compact form that presents the patterns to the user. The compact form could be seen as a “teaser” to look at the pattern in its original form. Remaining requirements and the implementation of such an application is left for future research.

# 7 Summary and conclusions

I have shown that it was possible to classify several patterns with *The Pattern Developers Categorization* and *The Pattern Users Categorization* using a compact form for easier interpretation of different patterns. This is a first draft and it will probably be altered many times before it can be used and accepted by all interested parties. Simplicity together with patience is what matters if a categorization ever will be used, which only will be a fact after many failures. To get many more suggestions on how a final outcome of a categorization of HCI patterns would look like, the classifying process and the categorizations have to be integrated and used in a development process. This is the first real attempt to create a categorization and to classify a large number of HCI patterns that others written. The first, but hopefully not the last.

# 8 References


[AIS77] Alexander, C.; Ishikawa, S.; Silverstein, M.; Jacobson, M.; Fiksdahl-King, I.; Angel, S. *A Pattern Language*. Oxford University Press. New York. 1977.

[BCC96] Beck, K.; Coplien, J. O.; Crocker, R,; Dominick, L.; Meszaros G.; Paulisch, F.; Vlissides, J. *Industrial Experience with Design Patterns*. http://www1.bell-labs.com/user/cope/Patterns/ICSE96/icse.html. 1996. Last visit: 2004-05-15.

[BER01] Bergin, J. *Pedagogical Patterns*. http://csis.pace.edu/~bergin/PedPat1.2.html. Pace University. 2001. Last visit: 2004-05-15.

[BOR01] Borchers, J. O. *A Pattern Approach to Interaction Desig*n. John Wiley & Sons, 2001.

[BRO88] Brown, C. M. Human-Computer Interface Design Guidelines (Human/Computer Interaction, No 5). Ablex Publishing. 1988.

[COO00] Cooper, J. W. *Java Design Pattern*s. Addison-Wesley. 2000.

[ERI00] T. Erickson. *Lingua Francas for Design: Sacred Places and Pattern Languages (paper for DIS 2000*). 2000.

[FLO01] Fetke, P.; Loos, P. *Zur Klassifikation von Patterns*. Organisatoren der Net.ObjectDays (Hrsg.): Net.ObjectDays 2001 – Tagungsband, 10 – 13. September 2001, Messekongresszentrum Erfurt. Erfurt 2001, ISBN 3-00-008419-3, S. 251-252.

[GAB96] Gabriel, R. *Design Patterns: Tales from the Software Community*. Oxford University Press Inc, USA. 1996.

[GHJ95] Gamma, E.; Helm, R.; Johnson, R.; Vlissides R. *Design Patterns: Elements of Reusable Object-Oriented Softwar*e. Addison-Wesley, Reading, MA, 1995.

[GRA99] Grand, M. *Patterns in Java, volume* 2. John Wiley & Sons. 1999.

[IPJ01] Ip, J. Use cases for Pattern Editing Tool (PET). http://www-cs-students.stanford.edu/~jip/use_cases.html. 2001. Last visit: 2004-05-15.

[JDP03] J2EE Design Patterns Repository. http://www.theserverside.com/patterns/index.jsp. TheServerSide.com. 2003. Last visit: 2004-05-15.

[MAT94] Mattisen R, Sipolt M.J. *The Object-Oriented enterprise – Making Corporate Information System Work*. McGraw-Hill. 1994.

[MAY97] Mayhew, D. J. *Principles and Guidelines in Software User Interface Design*. Pearson Education POD; 1st edition. 1997.

[MJO98] M. J Mahemoff and L. J. Johnston. *Pattern Language for Usability: An Investigation of Alternative Approache*s. In J. Tanaka, editor, APCHI '98 Proceedings, pages 25-31. IEEE Computer Society, Los Alamitos, CA. 1998.

[NOR88] D. A. Norman. *The Psychology of Everyday Things*. Basic Books, New York. 1988.

[PHP03] *Patterns Home Page*. http://hillside.net/patterns/. Hillside.net. 2003. Last visit: 2004-05-15.

[PPP03] *The Pedagogical Patterns Project*. http://www.pedagogicalpatterns.org. 2003. Last visit: 2004-05-15.

[PRE94] Preece, J. *Human-Computer Interaction*. Addison-Wesley Pub Co; 1st edition. 1994.

[SHN98] B. Shneiderman. *Designing the User Interfac*e, 3rd edition. Addison-Wesley, Reading, MA, 1998.

[SPH02] *Security Pattern Homepage*. http://www.securitypatterns.org. The IT Transfer Office (ITO), Darmstadt University of Technology, Department of Computer Science. 2002. Last visit: 2004-05-15.

[SSP95] Skogseid, I.; Spring, M. *Patterns for Human-Computer Interaction Studies of Principle Aggregation and Pattern Naming*. http://www.sis.pitt.edu/~spring/patterns/patterns.html. 1995.

[TID99] Tidwell, J. *Interaction Design Pattern*s. PLoP'98 Conference on Pattern Languages of Programming, Illinois, extended version including *Common Ground: A Pattern Language for Human-Computer Interface Design* at http://www.mit.edu/ ˜ jtidwell/interaction patterns.html. 1999. Last visit: 2004-05-15.

[VLI96] Vlissides, J. *Seven Habits of Successful Pattern Writers*. http://hillside.net/patterns/papers/7habits.html. 1996. Last visit: 2004-05-15.

[WTR00] M. van Welie; Traetteberg, H. *Interaction Patterns in User Interface*s. In: 7th. Pattern Languages of Programs Conference, Allerton Park Monticello, Illinois, USA, 13-16 August 2000.

[WVE00] M. van Welie; G. C. van der Veer; Eliëns, A. *Patterns as Tools for User Interface Desig*n. In: International Workshop on Tools for Working with Guidelines, Biarritz, France, 7-8 October 2000.

[YAN01] Yang, B. *E++: A pattern language for J2EE applications*. http://www.javaworld.com/jw-04-2001/jw-0420-eplus.html. JavaWorld. 2001. Last visit: 2004-05-15

# Appendix 1 – 73 HCI Patterns in Compact Form

In this thesis 73 HCI patterns have been the foundation for the result. Below all these patterns are listed in the compact form described in section 5. The meaning of the pattern can sometimes be hard to understand if the reader never has been exposed to it before in full form. To fully understand the HCI pattern I strongly recommend the reader to look at the originals. The reference to where the original patterns are to be found is given next to the pattern name.

**1. Narrative [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* In what form should the information be displayed to the user?
*When, How and Why:* When there is a need to convey information to the user; the information is closely interrelated but of diverse kinds, and there may be some subjectivity involved. Then, convey the information via natural language. A reason for this is that many people find it more pleasant to work with natural language than with symbolic representations. Another reason is that many people find narrative to be more pleasant than raw data or symbolic representations.

**High-density Information Display [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* In what form should the information be displayed to the user?
*When, How and Why:* When there is a need to convey lots of information, either of a homogeneous type or interrelated in some way, but all of roughly equivalent importance. Then, pack as much information as possible into one working surface as possible, following the precepts of good graphic design, with an organization that accurately reflects the underlying structure of information. A reason why this is a good solution is that the user often quickly want to find specific information. Another reason is that the user sometimes wants to grasp a big picture of the information fast.

**Status Display [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How can the artifact best show the state of information to the user?
*When, How and Why:* When the artifact must display state information that is likely to change over time, especially if that state information represents many variables. Then, 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. A reason for this is that users wants one place where they know they can find this state information.

**Form [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How should the artifact indicate what kind of information should be supplied, and the extent of it?
*When, How and Why:* When the user has to provide preformatted information, usually short (non-narrative) answers to questions. Then, provide appropriate ”blanks” to be filled in, which clearly and correctly indicate what information should be provided. Visually indicate those editable blanks consistently; such as with subtle changes in background color, so that the user can see at a glance what needs to be filled in. Label them with clear, short labels that use terminology familiar to the user; place the labels as close to the blanks as is reasonable. Arrange them all in an order that makes sense semantically, rather than simply grouping things by visual appearance. A reason for this is that it helps the user to see what is required and what is optional. Another reason is that it’s simple so there is not much need for directions, which the user seldom reads.

**Control Panel [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How can the artifact best present the actions that the user may take?
*When, How and Why:* When the artifact must provide a way for the user to either change its state, or command it to do something. Then, for each function or state variable that is part of the user’s mental model, choose one well-designed control that performs the function or displays the variable’s value; put them all together such that the most commonly-used controls are the most prominent. A reason for this is that the user wants one place where they know they can find the necessary controls, without having to hunt for them.

**WYSIWYG Editor [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How can the artifact best present what is being created, and what the user needs to do to create or change it?
*When, How and Why:* When the artifact is used as a tool or environment in which other artifacts may be created (particularly those with visual aspects). Then, always show the user an accurate and up-to-date representation of the artifact they are creating (”what you see is what you get”); allow the user to interact directly with it as they add to it, delete from it, modify it, and so on. A reason for this is that people already have the skills to draw, write, sculpt, arrange, etc. Another reason is that the user wants to see what they’re creating immediately, as they work with it; this instant feedback leads to a strong sense of engagement with the work.

**Composed Command [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How can the artifact best present the actions that the user may take?
*When, How and Why:* When the possible actions to be taken with the artifact can be expressed through commands, which can be composed from smaller parts, in a language-like syntax with precise and learnable rules; and the users are willing and able to learn that syntax. Then, provide a way for the user to directly enter the command, such as by speech or by typing it in. A reason for this is that experts often find linguistic commands to be more efficient than visual representations or direct manipulation. Another reason is that sometimes the available actions cannot, or should not, be expressed graphically (perhaps because there is an intractably large set of them).

**Navigable Spaces [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How can you present the content so that the user can explore it at their own pace, in a way which is comprehensible and engaging to the user?
*When, How and Why:* When the artifact contains a large amount of content – too much to be reasonable presented in a single view. This content can be organized into distinct conceptual spaces or working surfaces, which are semantically linked to each other, so that it's natural and meaningful to go from one to another. Then, create the illusion that the working surfaces are spaces, or places the user can "go" into and out of. A reason for this is that the user wants to know where they can (or should) go next, and how it's related to where they are now. Another reason is that it's delightful to explore a new place, where the user doesn't know what's "around the corner".

**Overview Beside Detail [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How can you present this large amount of content so that the user can explore it at their own pace, in a way which is comprehensible and engaging to the user?
*When, How and Why:* When the artifact contains a large amount of content – to much to be reasonably presented in a single view. This content may be cleanly partitioned into a top-level set of objects or categories, such as document names and document content, containers and their contents, or objects and their properties; alternatively, the content may be a large, continuos, very detailed data set, in which users may have specific areas of interest. Then, show the whole set of objects, or the whole undetailed set, in one part of the display area, to act as an overview of the content. When the user selects a single object, category, or area of interest within that overview, immediately show it's related content – its detail – in the remaining space. As the user changes the selection, update the detail area to always reflect the current selection. A reason for this is that user want to move between the detail views quickly, e.g. to compare and contrast. Another reason for this is that it lets user concentrate on one object, category, or area of interest at a time.

**Step-by-Step Instructions [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How can the artifact unfold the possible actions to the user in a way that does not overwhelm or confuse them, but instead guides them to a successful task completion?
*When, How and Why:* When a user needs to perform a complex task, with limited time, knowledge, attention, or space. Alternatively, the nature of the task is step-by-step, and it's meaningless to show all the action possibilities at once. Then, walk the user through the task one step at a time, giving very clear instructions at each step. A reason for this is that the user doesn't always want, or need, to understand all the details of what they are doing. Another reason is that a user who is afraid of doing something wrong may prefer that the actions they have to perform be explicitly spelled out for them.

**Small Groups of Related Things [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How should the items or actions be organized?
*When, How and Why:* When there are many items or actions to show the user, some of which are more closely related to each other than other things. Then, group the closely related things together, nesting them in a hierarchy of groups if needed. A reason for this is that large, undifferentiated masses of things can be intimidating and difficult to figure out, especially to someone seeing them for the first time. Another reason is that people naturally assume semantic coherence where there is visual coherence.

**Hierarchical Set [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How should the information be organized?
*When, How and Why:* When there are many things to show the user, and they are interrelated in a hierarchy (or can be made to appear that way). Then, Show the data in a tree-like structure. A reason for this is that the user should see the structure of the data. Another reason is that hierarchies are easy to understand.

**Tabular Set[TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How should the information be organized?
*When, How and Why:* When there are many homogeneous things to show the user, each of which has similar additional information or subparts. Then show the data in a table structure. A reason for this is that the user should see the structure of the data. Another reason is that the user wants to easily look up a specific piece of information.

**Chart or Graph [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How should the information be organized?
*When, How and Why:* When there is a lot of homogeneous data to show the user, possibly in multiple data sets. Then, show the data plotted against time or some other variable. Plot it together with other variables for further comparison. A reason for this is that the user wants to get a ”big picture” quickly. Another reason is that graphs facilitate easy comparison of values to each other.

**Optional Detail On Demand [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* When should these usually unneeded items be presented to the user, and how?
*When, How and Why:* When a large percentage of the available information or actions (termed ”items” for the rest of this pattern) can be considered details, and are unneeded most of the time. Then, up front, show the user that which is most important and most likely to get used. Details and further options which won’t be needed most of the time – say 20% or less of expected uses – can be hidden in a separate space or working surface (another dialog, another piece of paper, behind a blank panel). A reason for this is that the user may be confused or overwhelmed if they are presented with lots of items at once. Another reason is that sometimes there isn’t enough space to show all the possible items.

**Disabled Irrelevant Things [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How can the artifact steer the user away from actions that cannot or should not be taken, while still maintaining visual calm and stability?
*When, How and Why:* When information or actions that are normally useful become temporarily irrelevant. Then, disable all the things that have become irrelevant. A reason for this is that the artifact should present items in such a way as to allow the user to form a correct mental model of its underlying ideas (for information) or action states (for actions). Another reason is that when the users can trust the artifact to not let them into invalid actions, they will feel more secure about exploring it and trying new things.

**Pointer Shows Affordance [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How can the artifact indicate that a visual entity represents an action that user may take?
*When, How and Why:* When the artifact contains a visual pointer, or ”virtual fingertip” (mouse or pen point, for instance) that is the focal point for the user’s interaction with the artifact. Then, change the affordance of the thing as the pointer moves over it. A reason for this is that static visual affordances aren’t always enough to indicate the presence of a manipulable control, especially when space is tight or when an elegant-looking graphic design is of paramount importance. Another reason is that multiple affordances can be more effective than one, if they work together properly.

**Short Description [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How should the artifact present additional content. In the form of clarifying data or explanations of possible actions, to the user that needs it?
*When, How and Why:* When the artifact contains a visual pointer, or ”virtual fingertip” (mouse or pen point, for instance) that is the focal point for the user’s interaction with the artifact. Then, show a short (one sentence or shorter) description of a thing, in close and/or temporal proximity to the thing itself. A reason for this is that users generally don’t want to leave the artifact and go somewhere else for help, such as a manual; this usually breaks one’s concentration and costs too much time. Another reason is that there isn’t room to put static descriptive text into the artifact, or visual elegance precludes doing so.

**Sovereign Posture [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How should this artifact relate spatially to other artifacts that might share its space, and how can it best use the space it has?
*When, How and Why:* When the artifact will be heavily used, occupying the user’s full attention, and the user is willing to invest time and effort to learn it. Then, allow the artifact to take up all the space it needs to get the job done efficiently and gracefully. However, don’t take up too much space or time explaining what things are and what needs to be done, since the amount of time spent learning it will be trivial compared to the time spent using it as an experienced user. A reason for this is that the user wants full access to everything at once, even if the artifact is complex. Another reason is that the user will be irritated by small inefficiencies and too much hand holding.

**Helper Posture [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How should this artifact relate spatially to other artifacts that might share its space, and how can it best use the space it has?
*When, How and Why:* When the activity supported by the artifact is secondary to other activities, but will occasionally require the user's full attention for a short time. Then, use as much space as needed to make it comprehensible, but no more; focus tightly on the activity by excluding all but the commonest actions, information, etc. from the top level, but let those common ones take whatever space they need. A reason for this is that the user is principally doing something else, and this shouldn't interfere with it. Another reason is that the user has little or no incentive to spend time learning the artifact, so its learning curve should be as short as possible.

**Background Posture [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How should this artifact relate spatially to other artifacts that might share its space, and how can it best use the space it has?
*When, How and Why:* When the activity supported by the artifact is secondary to other activities, and will never need more than a little of the user's attention; but it should stay around for those times the user does need it. Then, make the artifact small, relative to the other primary activities going on at the same time, and keep it unobtrusive. A reason for this is that the user will not bother to expend much mental effort to try to learn or use the artifact, because the activity's priority is low; its learning curve should be extremely short or nil. Another reason is that the user wants the artifact to stay around, in case it's needed.

**Central Working Surface [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How should the artifact's working surfaces be organized?
*When, How and Why:* When the artifact is composed of multiple working surfaces, and is centered on one particular activity. Then, create one working surface where the artifact's major functions are collected together; if most of the work can actually be done there, so much the better, Secondary functionality may be placed on secondary working surfaces, which can be easily reached from the central one and which allow the user to easily get back to the central one. A reason for this is that the user should have one predictable place where most of their work with the artifact gets done. Another reason is if the artifact is complex, the user may get disoriented as they interact with its various working surfaces; they may want a familiar "home base" to go to.

**Tiled Working Surfaces [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How should the artifact's working surfaces be organized?
*When, How and Why:* When the artifact displays anything visual, and can be split up into multiple working surfaces; there is enough space to show all those working surfaces at once. Then, place the working surfaces together in a plane, such that do not obscure each other, and show the whole thing to the user. A reason for this is that the user wants easy access to many working surfaces. Another reason is that some users don't want to (or can't) manipulate multiple working surfaces to see them all.

**Stack of Working Surfaces [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How should the artifact's working surfaces be organized?
*When, How and Why:* When the artifact displays anything visual, and can be split up into multiple working surfaces. Then, stack the surfaces together. Label each surface with a unique and recognizable name or icon (or let the user pick the label), and visually cluster those labels together near the stack. A reason for this is that the user wants easy access to many working surfaces. Another reason is that each surface needs, or could at least use, all the space available.

**Pile of Working Surfaces [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How should the artifact's working surfaces be organized?
*When, How and Why:* When the artifact displays anything visual, and can be split up into multiple working surfaces. Then, stack the surfaces loosely so that they obscure each other most of the time, but so that one or more surfaces of the user's choosing can be on top. A reason for this is that the user can identify them by name or icon so that they can be brought to the top when needed. Another reason is that the user wants easy access to many working surfaces.

**Map of Navigable Spaces [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How can the artifact help a user navigate effectively and remain oriented?
*When, How and Why:* When the artifact (or its content) can be organized into distinct spaces or working surfaces which are semantically linked to each other, so that it's natural to go from one to another. Then, provide a map or diagram of the Navigable Spaces relevant to the artifact. Organize it appropriately, and put it where the user can easily get at it; if possible, let it be seen side-by-side with what the user is doing. A reason for this is that the user should know where they are at any time. Another reason is that the user wants to know where they can go next, and how to get there.

**Clear Entry Points [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How does the user know where to start?
*When, How and Why:* When the artifact is organized as a set of Navigable Spaces; in particular, the artifact itself is large or contains a large amount of content. Then, provide a small set of well defined, clearly named entry points to the network of Navigable Spaces. A reason for this is that a complex artifact can be very disorienting for the user. Another reason is that if a user starts at a randomly chosen place in Navigable Spaces network, they probably won't have enough contextual information to figure out how best to do what they need to do.

**Color-Coded Sections [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How can an artifact both give users a sense of place, and also tell them where they are, within a large network of spaces?
*When, How and Why:* When the artifact is made up of a large number of Navigable Spaces, which is organized into a small number of major subsections. Then, use color to identify the major sections of the artifact. Pick one color per major section, and use it on every space or working surface within that section. A reason for this is that the user has a need to know approximately where he or she is, at all times. Another reason is that a text label saying what section the user is in may be ugly or unnoticeable, and may be far more incomprehensible than text.

**Go Back One Step [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How can the artifact make navigation easy, convenient, and psychologically safe for the user?
*When, How and Why:* When the artifact allows a user to move through spaces. Then, provide a way to step backward to the previous space or state. If possible, let the user step backwards multiple times in a row, thus allowing them to backtrack as far as they want. A reason for this is that users tend to explore a navigable artifact in a tree-like fashion, going down paths that look interesting, then back up out of them, then down in another path. Another reason is that the user may want to temporarily look back at the previous space or state they were in.

**Go Back to a Safe Place [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How can the artifact make navigation easy, convenient, and psychologically safe for the user?
*When, How and Why:* When the artifact allows a user to move through spaces. Then, provide a way to go back to a checkpoint of the user's choice. A reason for this is that a user may forget where they were, if they stop using the artifact while they're in the middle of something and don't get back to it for a while. Another reason is that if the user gets into a space or state that they don't want to be in, they will want to get out of it in a safe and predictable way.

**Convenient Environment Actions [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How should the artifact present actions that affect the existence or state of the whole artifact?
*When, How and Why:* When the user can take actions that affect the existence or state of the whole artifact. Then, group these actions together, label them with words or pictures whose meanings are unmistakable, and put them where the user can easily find them regardless of the current state of the artifact. A reason for this is that the user should know exactly how to stop or leave this artifact at any time. Another reason is that the user should know what other actions are available.

**Localized Object Actions [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How should the artifact present the actions that may be taken on real or virtual objects that are in the artifact?
*When, How and Why:* When the artifact contains multiple real or virtual objects, such as files, or CDs, or car windows. Then, group object actions together, even more so than for Convenient Environment Actions, and spatially localize them to the object. Separate them visually from any other kinds of actions, to avoid potential confusion. A reason for this is that the user should know what actions they can take that affect the chosen object(s). Another reason is that the user may already know what they have to do, but they need to find the corresponding action.

**Actions for Multiple Objects [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How can the artifact make repetitive tasks easier for the user?
*When, How and Why:* When the artifact contains multiple real or virtual objects, such as files, or CDs, or car windows. There are actions to be performed on those objects, and users are likely to want to perform actions on two or more objects at one time. Then, allow the action to be performed ”in parallel” across a set of user-selected objects. Make it easy to put that collection together. A reason for this is that people aren’t good at repetitive tasks; machines are. Another reason is that users don’t want to spend time performing the same action to every single object separately.

**Choice from a Small Set [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How should the artifact indicate what kind of information should be supplied?
*When, How and Why:* When the artifact shows, or allows the user to set, a value which is one out of a small set of possible values (10 or fewer). Then, show all the possible choices up front, show clearly which choice(s) have been made, and indicate unequivocally whether one or several values can be chosen. A reason for this is that the user should see all the possible values, to put the actual value in context. Another reason is that small number of things can be taken stock of quickly, and don’t take up much space.

**Choice from a Large Set [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How should the artifact indicate what kind of information should be supplied?
*When, How and Why:* When the artifact shows, or allows the user to set, a value which is one out of a large set of possible values (more than 10). Then, clearly show the selected value up front; organize the set of possible values, but hide them nearby if they take up too much space. A reason for this is that the user should be able to find the value they want quickly. Another reason is that large numbers of things take a long time to read and take up a lot of space.

**Sliding Scale [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How should the artifact indicate what kind of information should be supplied?
*When, How and Why:* When the value to be shown or set is scalar (not necessarily numeric), appears to the user to be continuos, and is bounded at both ends. Then, show the range of values visually, as with a line or arc; show the current value as a location in that line or arc, and if the value is settable by the user, make that location directly changeable by the user, as with a slider or knob – or by simply touching or clicking on the desired value. A reason for this is that it's often easier to directly manipulate a value than to enter it via text. Another reason is that the user should see all the possible values, to put the actual value in context.

**Editable Collection [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How should the artifact indicate what the user is supposed to do with an ordered set of things?
*When, How and Why:* When the user should build or modify an ordered set of things, possibly (but not necessarily) chosen from a larger set. Then, show the collection to the user, along with obvious ways to remove or change the position of each item. To add an item, make it eminently clear whether the user should obtain the item before or after the "add" command or gesture. A reason for this is that the user should know what the collection currently has in it.

**Forgiving Text Entry [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How does the artifact indicate what kind of information should be supplied?
*When, How and Why:* When the user should enter information. Then, allow the user to enter text in any recognizable format for that context. Be forgiving of formatting idiosyncrasies, formatting mistakes, or recognizable 'typos". A reason for this is that too much constraint can be annoying to the user. Another reason is that both computers and humans – depending on who's interpreting the entered information – are good at figuring out what a given text entry is intended to be, if it even vaguely resembles some accepted format.

**Structured Text Entry [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How does the artifact indicate what kind of information should be supplied?
*When, How and Why:* When the user should enter information, as on a Form, but that information must be in a very specific format. Then, rather than letting a user enter information into a blank and featureless textfield, put structure into that textfield. Divide it into multiple fields with different relative sizes, for instance, or superimpose a faint visual design on it (like dividers or decimal points). A reason for this is that the user doesn't need the cognitive burden of figuring out what specific format is acceptable. Another reason is that if the user sees exactly what is expected of them, they don't have to be uncertain about what they're entering into the text field.

**Toolbox [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How should the artifact present the actions that the user may take?
*When, How and Why:* When the artifact supports the creation of other artifacts, as in WYSIWYG Editor. Then, keep tools together, put them physically close to the user's working surface, and make sure they are distinct from other actions that the user may take. If the tools can be represented well as icons and not words, use icons; they usually represent visual objects or modes anyway. The reason for this is that tools used to create things are different in kind from ordinary actions performed upon existing objects. Another reason is that an experienced user needs the tools ready to hand.

**User Preferences [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How does the artifact present the actions that the user may take?
*When, How and Why:* When the artifact will be used by people with differing abilities, cultures, and tastes. Then, provide a place or working surface where the users can pick their own settings for things like language, fonts, icons, color schemes, and use of sounds. Allow users to save those preferences, so that they don't have to spend time setting them up again. A reason for this is that the user may want some degree of aesthetic control over the artifact. Another reason is that users get a sense of ownership and control over an artifact by modifying it.

**Personal Object Space [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How should items that the user needs ready access to be organized?
*When, How and Why:* When there are many things that the user needs ready access to, such as working surfaces, documents, objects, or tools. Then, allow users to place things where they want, at least in one dimension but preferably in two. A reason for this is that the user should be able to arrange things in a way that works best for them, since they know more about how they work than the artifact's designer does. Another reason is that the user wants a sense of ownership and control over the artifact, and substantial customization contributes to that sense.

**Scripted Action Sequence [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How can the artifact make repetitive tasks easier for the user?
*When, How and Why:* When the user needs to perform the same sequence of actions over and over again, with little or no variability. This often happens in artifacts, which presents a very wide spectrum of actions to the user. Then, provide a way for the user to "record" a sequence of actions of their choice, and a way to easily "play them back" at any time. A reason for this is that people aren't good at repetitive tasks; machines are. Another reason is that artifact designers can't build in shortcuts for every possible repetitive action sequence, partly because there are to many possible combinations, and partly because the repeated sequences vary to much from one individual user to the next.

**User's Annotations [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How can the artifact help preserve the user's hard-won understanding from one use session to the next?
*When, How and Why:* When the artifact is complex and difficult to learn, but will be used again by the same user or by others. Then, support ways for the users to add their own comments and other annotations to the artifact. Allow the users to place those annotations physically close to where they are needed, and if possible, allow for simple drawings in addition to text. A reason for this is that users know better what's memorable to them than the artifact does (or the artifact's designer). Another reason is that the proper spatial placement of help text can increase its effectiveness.

**Bookmarks [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How can the artifact support the users need to navigate through it in ways not directly supported by the artifact's structure?
*When, How and Why:* When the artifact is large or complex, and allows the user to move freely through it. Then, let the user make a record of their points of interest, so that they can easily go back to them later. A reason for this is that a user may want to keep track of the places that are most interesting or most useful, for future reference. Another reason is that a user in "browse mode" may quickly visit a place, decide it's worth coming back to later, and keep going in a different direction; they need to keep track of where they want to return.

**Repeated Framework [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How should the content be presented in a unified and consistent way, so that the user can easily navigate through it and quickly become familiar with it?
*When, How and Why:* When the artifact contains a medium or large amount of content through which the user will navigate; that content is subdivided into many pages or working surfaces. Then, design a simple, flexible visual framework for the content, then repeat it on every page or working surface; position the content within that framework, allowing the form of the content to vary as needed. Put functional elements, such as headers and common controls, in the same place within the framework on each page; these will usually be at the edges. A reason for this is that the artifact needs to be visually unified, to look like at all of one piece. This may be done for commercial reasons, such as to establish a corporate identity, but it also improves the user's experience by helping maintain "visual momentum" as the user moves through the artifact. Another reason is that different content often demands different visual treatment, to make its structure and meaning clear.

**Good Defaults [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How does the artifact indicate what kind of information should be supplied?
*When, How and Why:* When the user should fill in information on a Form (or change settings via a Control Panel), and some of the data fields can be given reasonable default values. Then, supply reasonable default values for the fields in question. Show these defaults to the user, so that they aren't required to fill them in. A reason for this is that the user may have no clue what kind of value to supply, from the given context. Another reason is that the user may be perfectly happy with the default behavior or values, with no desire to change it; but they may want to know what the default values are.

**Remembered State [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How can the artifact help save the user time and effort?
*When, How and Why:* When the artifact lets users enter information (e.g. Form), set its state in various ways (as with Control Panel or WYSIWYG Editor), or customize it; and the artifact is likely to be used again soon by the same user. Then, design the artifact so that it can remember its state from session to session. If multiple users are likely to use it, make sure the state is saved on a per-user basis. A reason for this is that users don't want to redo their customizations from scratch every time they use the artifact. Another reason is that things in real life do this automatically – your car seat stays reclined to the same place, bookmarks stay in books, etc.

**Interaction History [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* Should the artifact keep track of what the user does with it? If so, how?
*When, How and Why:* When the user performs a sequence of actions with the artifact, or navigates through it. Then, record the sequence of interactions as a "history". Keep track of enough detail to make the actions repeatable, scriptable, or even undoable, if possible. A reason for this is that people are forgetful of tedious details; users are not likely to remember just what they've recently done with the artifact, and computers are better at it than people are. Another reason is that the user may need to know exactly what they've just done, so they can undo their work or backtrack.

**Progress Indicator [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How can the artifact show its current state to the user, so that the user can best understand what is going on and act on that knowledge?
*When, How and Why:* When a time-consuming process is going on, the results of which is of interest to the user. Then, show the user a status display of some kind, indicating how far along the process is in real time. If the expected end time is known, or some other relevant quantity (such as the size of a file being downloaded), then always show what proportion of the process has been finished so far, so the user can estimate how much time is left. A reason for this is that the user wants to know how long they have to wait for the process to end. Another reason is that the user wants to know that progress is actually being made, and that the process hasn't just "hung".

**Important Message [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How should the artifact convey a message that has to catch the user's attention immediately?
*When, How and Why:* When the user must be informed of something immediately during usage of an artifact. Then, interrupt whatever the user is doing with the message, using both sight and sound if possible. Communicate the message in clear, brief language that can be understood immediately, and provide information on how to remedy the situation, unless the cultural meaning of a non-lingual message is so strong that it cannot be mistaken (like a fire alarm). A reason for this is that the user is probably paying attention to something else at the time the event happens. Another reason is that the user may be short of time or under stress, possibly as a result of the message itself, so they won't be in a good position to stop and think about how to react.

**Reality Check [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How can the artifact protect itself and the user from an action that can have destructive and non obvious side effects, while allowing the user to have the final say over whether or not an action is performed?
*When, How and Why:* When the user is performing an action which may have destructive or non obvious side effects, especially if that action isn't reversible. Then, before the action is performed, tell the user what the side effects of the action will be, and ask the user to confirm that that's what they really want to do. Don't simply parrot back the action request – this won't tell the user anything they don't already know, unless the action request was accidental in the first place. Instead give them an intelligent analysis of what the action may do, in case they did not anticipate the potential side effects. A reason for this is that the user may be deeply confused by unexpected side effects to their actions (and may not even be able to connect them to the action that caused them), leading to a mixed-up mental model and distrust of the artifact. Another reason is that the user may understand the ramifications of the action better than the artifact does; they should have the ultimate choice about whether or not to perform the action.

**Demonstration [TID99]**
*Author(s):* Jennifer Tidwell
*Problem:* How can the user learn how to use the artifact?
*When, How and Why:* When a user needs to understand how to do something complex, usually either performing a task or creating something freeform (as with a WYSIWYG Editor). Then, demonstrate how to do it. Show a video clip, or drive the software to actually do the task (or something representative of a typical usage) in front of the user. A reason for this is that the task will be easy to remember once the user sees how to do it. Another reason is that the user doesn't have the time or inclination to read written help, such as a manual.

**Wizard [WTR00]**
*Author(s):* Martijn van Welie
*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.
*When, How and Why:* When a non-expert user needs to perform an infrequent complex task consisting of several subtasks where decisions need to be made in each subtask. The number of subtasks must be small e.g. typically between three and ten. Then, 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. A reason for this is that the user wants to reach the overall goal but may not be familiar or interested in the steps that need to be performed. Another reason is that in order to reach the goal, several steps need to be taken but the exact steps required may vary because of decisions made in previous steps.

**Grid Layout [WTR00]**
*Author(s):* Martijn van Welie
*Problem:* The user needs to quickly understand information and take action depending on that information.
*When, How and Why:* When there are several information objects that are presented and arranged spatially on a limited area. Typically in the design of dialog screens, forms and web pages. Then, arrange all objects in a grid using the minimal number of rows and columns, making the cells as large as possible. A reason for this is that the user needs to see many objects but want to see them in a clear organized way. Another reason is that the objects are often related and can be grouped conceptually.

**Progress [WTR00]**
*Author(s):* Martijn van Welie
*Problem:* The user wants to know whether or not the operation is still being performed as well as how much longer the user will need to wait.
*When, How and Why:* When there are a system task that tale a long time (typically more than a few seconds) and must be completed before the next task can be started. Then, show that the application is still working and give an indication of the progress. A reason for this is that the users may not be familiar with the complexity of the task. Another reason is that in some situations the user might want to terminate the operation because it will take too long time.

**Shield [WTR00]**
*Author(s):* Martijn van Welie
*Problem:* The user may accidentally select a function that has irreversible (side) effects.
*When, How and Why:* When the user might not be aware of the irreversible side effects that a function might have. Then, protect the user by inserting a shield. An extra layer to the function so that the user is asked to confirm its intent with the default answer being the safe option. A reason for this is that the user needs to be protected but normal operation should not be changed. Another reason is that the user may be striving for speed while trying to avoid making mistakes.

**Preferences [WTR00]**
*Author(s):* Martijn van Welie
*Problem:* Each user is different and prefers to do things slightly different.
*When, How and Why:* When the application is very complex and many of its functions can be tuned to the user's preference. Not enough is known about the user's preferences in order to assume defaults that will suit all users. The potential users can range from novice to expert. Then, allow the user to adjust their preferences. Instead of forcing a single profile for each user, allow the users to change their preferences. A reason for this is that the user can improve satisfaction by streamlining usage. Another reason is that this gives the user the possibility to only change a subset of all options.

**Contextual Menu [WTR00]**
*Author(s):* Martijn van Welie
*Problem:* At any point in time, users need to know what their possibilities are in order to decide what to do.
*When, How and Why:* When an application contains a lot of functionality and the user needs to know her possibilities at any point during use. The user is typically a novice or casual user and the functions are used infrequently. Then, put the choices in a menu. Show a list of all functions that are available in the current context and make the functions available in one action. A reason for this is that the number of possibilities may be large and the user needs to locate the desired one. Another reason is that this is integrating learning and using, while not hindering the expert.

**Focus [WTR00]**
*Author(s):* Martijn van Welie
*Problem:* Users want to quickly know information about an object they see and possibly modify the object.
*When, How and Why:* When an application with several visual objects are manipulated, typically drawing packages or browsing tools. Then, introduce a focus in the application. The focus always belongs to an object present in the interface. The focus i.e. object the user is working on determines the context of the available functionality. The focus must be visually shown to the user for example by changing the colour or by drawing a rectangle around it. When an object has the focus it becomes the target for all the functionality that is relevant for the object. A reason for this is that many objects can be visible but the user usually works on one object at a time. Another reason is that the user both wants an overview of the set of objects and details on attributes and available functions on the objects.

**Unambigious Format [WTR00]**
*Author(s):* Martijn van Welie
*Problem:* The user needs to supply the application with data but may be unfamiliar with which data is required or what syntax to use.
*When, How and Why:* When there is a system where structured data must be entered. Data such as dates, room numbers, social security numbers or serial numbers are usually structured. The exact syntax for such data may vary per country or product. Then, only allow the user to enter data in the correct syntax. A reason for this is that data entered using an unexpected syntax can not be used by the application. Another reason is that users often strive for speed but want their entries to be correct at the same time.

**Navigating between Spaces [WTR00]**
*Author(s):* Martijn van Welie
*Problem:* The user needs to access an amount of information which cannot be put on the available space.
*When, How and Why:* When we have a system with a lot of states, functionality and objects, that are relevant only in groups. Then, show the information in several spaces and allow the user to navigate between them. Group the information elements in separate spaces. Allow the user only to select one space at the time. A reason for this is that large amount of data require a lot of space but the available display space is limited. Another reason is that large amounts of data are usually not unrelated and can be divided into categories that match the user's conceptual model of the data.

**Like in the real world... [WTR00]**
*Author(s):* Martijn van Welie
*Problem:* The user needs to know how to control an object in the interface which resembles an object the user knows from the real world.
*When, How and Why:* When we are dealing with an application that use real world metaphors and direct manipulation in the interfaces. The objects in the interfaces resemble real world objects and interaction is suggestion to resemble real world interaction. Then, match the input device and the widget used. Use a combination of widget and input device that match in terms of the possible movements. Movements include the degrees of freedom and types of feedback. A reason for this is that interaction is done with input devices but each device has a limited set of manipulation possibilities such as movement, rotation, force feedback, and degrees of freedom. Another reason is the way objects in the real world are manipulated. That is creating expectations about how the interaction will take place.

**Preview [WTR00]**
*Author(s):* Martijn van Welie
*Problem:* The user is looking for an item in a small set and tries to find the item by browsing the set.
*When, How and Why:* When we have an application where the user needs to find an item e.g. a file, a presentation, video clip, or an image. In the application a visual or auditory search criterion is more effective but the index of the set is not audiovisual (e.g. a text label). Then, allow the user to preview the item. Show a representative in-place preview of one or more items. The preview can use fewer resources such as screen, loading time, or quality than the original. A reason for this is that the user might not be able to identify the item by just the index name. Another reason is that the user might be looking for an item but needs the index name as a search result for use in other tasks.

**Favourites [WTR00]**
*Author(s):* Martijn van Welie, Hallvard Traetteberg
*Problem:* The user needs to find a regularly used item in a large set of items.
*When, How and Why:* When the user is looking for an item that is contained in a large set of items. The item is of importance and the user requires it regularly. Items are typically files, colours, web pages or database records and are part of large collections, respectively a file system, the web or a database. Then, allow the user to use favourites that point to the items of choice. Introduce a possibility to create and remove favourites. A favourite is a label that points directly to the particular item. A reason for this is that the user knows of the item's existence and uses it regularly, hence the items need to be at hand. Another reason is that the user is interested in the item but may use the item only for a short period of time.

**Command Area [WTR00]**
*Author(s):* Martijn van Welie
*Problem:* The user needs to know where to find the possible commands and how to activate them.
*When, How and Why:* When the functionality in an application needs to made known to the user. Then, put the commands in a specific recognizable area. Reserve an area on the screen for access to functions. A reason for this is that some functions are more used than other functions. Another reason is that the available screen estate is limited and the main working area should be kept as large as possible.

**Container Navigation [WTR00]**
*Author(s):* Martijn van Welie, Hallvard Traetteberg
*Problem:* The user needs to find an item in a collection of containers
*When, How and Why:* When an application contain aggregated data, which the user must browse through. Quite often, the user wants to invoke a function taking one of the parts as input parameter. Then, split up the screen in three areas showing the containers and the final selection. Split a window into three parts, one for the viewing a collection of containers, one for viewing a container, and one for viewing individual items. The selection of a container should determine the content of the container pane, and the selected item should determine the content of the item pane. A reason for this is that the number of items is large but only a portion of the items needs to be visible at a particular moment.

**Setting Attributes [WTR00]**
*Author(s):* Hallvard Traetteberg
*Problem:* Users want to see the attributes of the object they are working on and additionally they need to know how to modify them.
*When, How and Why:* When a document in an application can hold many different objects with many different attributes. Then, create special widgets that show the attribute value. Frame the document pane with dialog elements for showing the various attribute types. Include only the most common/generic ones and let the user customize the available elements. Provide access through these to dialog elements for setting the same attribute types. A reason for this is that setting attributes requires different controls than when visualizing them. Another reason is that objects are often of several types and have different sets of attributes.

**Warning [WTR00]**
*Author(s):* Martijn van Welie, Hallvard Traetteberg, David Kane
*Problem:* The user may unintentionally cause a problem situation which needs to be resolved.
*When, How and Why:* When situations occur where the user performs an action that may unintentionally lead to a problem. Then, warn the user before continuing the task and give the user the chance to abort the tasks. A reason for this is that important work may be lost if the action is fully completed. Another reason is that some actions are difficult or impossible to recover from.

**Hinting [WTR00]**
*Author(s):* Martijn van Welie
*Problem:* The user needs to know how to select functions.
*When, How and Why:* When the functionality in an application is accessible in more than one way, e.g. through menus, using keyboard shortcuts, or through toolbars. Then, give the user hints for other ways to access the same function. A reason for this is that the user needs some way of discovering and learning these alternatives and possibly more efficient way, in a non-obtrusive way.

**Mode Cursor [WTR00]**
*Author(s):* Martijn van Welie
*Problem:* The user is creating or modifying an object and needs to know which edit function is selected.
*When, How and Why:* When a user selects a tool/function in a direct manipulation application which takes the application into a special mode/state. Thereafter the user works on an object in this state. These kinds of applications often contain many functions to create or modify objects and thereby the user needs to select a tool first. Then, show the interface state in the cursor. The interface state changes many times during interaction, for instance when a function is selected or when an action such as dragging is performed. Therefore, show the current state to the user by changing the cursor. A reason for this is that the user needs immediate feedback on which function was selected, i.e. which mode/state the system is in. Another reason is that while completing a function many intermediate states may also need to be shown.

**List Browser [WTR00]**
*Author(s):* Martijn van Welie, Hallvard Traetteberg
*Problem:* The user needs to browse or process several items out of a set or list.
*When, How and Why:* When the user has to go through a list of items in an application. For example, when reading news items on a news web site or when browsing through the results of a database query. The user typically selects an initial item out of the list and then wants to move on to other articles, typically the next one or the previous one. In general, the user wants to read several items of the list in the order, backwards or forwards. Then, find a natural ordering and allow the user to navigate directly from one item to the next and back. A reason for this is that the user wants to see an overview of the set/list and at least one item, but the screen space may not be sufficient for both. Another reason is that the user needs to be able to select an individual item as well as to process several items.

**Continous Filter [WTR00]**
*Author(s):* Martijn van Welie, Karri-Pekka Laakso
*Problem:* The user needs to find an item in an ordered set.
*When, How and Why:* When the goal is to allow the user to dynamically narrow the search depending on the immediate feedback given by the continious filter. Then, provide a filter component with which the user can in real time filter only the items in the data that are of his interest. A reason for this is that the user is searching for an item in a large ordered set and may not be familiar with the exact item, nor is the user sure the item exists.

# Appendix 2 – The Classification

The classification of the patterns presented in the compact form in appendix 1 is presented below. I have chosen to use the sign X to mark that the attribute apply to that pattern and an O indicates that the attribute doesn't apply. For more information about the categories and the classification process see section 5.

**Table 6: The Pattern Developers Categorization, usability**

| Pattern | Visibility | Constraints | Affordance | Mappings | Feedback |
|---|---|---|---|---|---|
| Narrative | O | O | O | X | O |
| High-density Information Display | X | O | O | O | O |
| Status Display | X | X | O | O | O |
| Form | O | O | O | X | O |
| Control Panel, Command Area | X | X | O | O | O |
| WYSIWYG Editor | O | O | O | X | X |
| Composed Command | O | O | O | O | O |
| Navigable Spaces | O | X | O | O | O |
| Overview Beside Detail | X | X | O | O | O |
| Step-by-Step Instructions | O | X | O | X | O |
| Small Groups of Related Things | X | X | O | O | O |
| Hierarchical Set | X | O | O | X | O |
| Tabular Set | X | X | O | X | O |
| Chart or Graph | X | O | O | X | O |
| Optional Detail On Demand | X | O | O | O | O |
| Disabled Irrelevant Things | O | X | O | O | O |
| Pointer Shows Affordance, Mode Cursor | O | O | X | O | X |
| Short Description | O | O | O | O | O |
| Sovereign Posture | O | X | O | O | O |
| Helper Posture | O | X | O | O | O |
| Background Posture | O | X | O | O | O |
| Central Working Surface | O | X | O | O | O |
| Tiled Working Surfaces | O | X | O | O | O |
| Stack of Working Surfaces | O | X | O | O | O |
| Pile of Working Surfaces | O | X | O | O | O |
| Map of Navigable Spaces | O | O | O | X | X |
| Clear Entry Points | O | X | O | O | O |
| Color-Coded Sections | X | X | O | O | O |
| Go Back One Step | O | O | O | O | O |
| Go Back to a Safe Place | O | O | O | O | O |
| Convenient Environment Actions | X | X | O | X | O |
| Localized Object Actions | X | X | O | X | O |
| Actions for Multiple Objects | O | O | O | O | O |
| Choice from a Small Set | X | O | O | O | X |
| Choice from a Large Set | X | X | O | O | X |
| Sliding Scale | O | X | O | X | X |
| Editable Collection | O | O | O | O | X |
| Forgiving Text Entry | O | O | O | O | O |
| Structured Text Entry | X | X | O | O | O |
| Toolbox | X | X | O | O | O |
| User Preferences, Preferences | O | X | O | O | O |
| Personal Object Space | X | O | O | X | O |
| Scripted Action Sequence | O | O | O | O | O |

| Pattern | Visibility | Constraints | Affordance | Mappings | Feedback |
|---|---|---|---|---|---|
| User's Annotations | O | O | O | X | O |
| Bookmarks, Favourites | O | O | O | X | O |
| Repeated Framework | X | X | O | O | O |
| Good Defaults | O | X | O | O | O |
| Remembered State | O | O | O | O | O |
| Interaction History | O | O | O | O | O |
| Progress Indicator, Progress | O | O | O | O | X |
| Important Message | X | O | O | O | O |
| Reality Check, Shield, Warning | O | O | O | O | X |
| Demonstration | O | O | O | O | O |
| Wizard | O | X | O | O | O |
| Grid Layout | X | X | O | O | O |
| Contextual Menu | O | X | O | O | O |
| Focus | X | O | O | O | O |
| Unambigious Format | O | O | O | O | O |
| Navigating between Spaces | X | X | O | O | O |
| Like in the real world? | O | O | O | X | O |
| Preview | X | O | O | O | O |
| Container Navigation | X | O | O | O | O |
| Setting Attributes | X | O | O | O | O |
| Hinting | O | O | O | O | O |
| List Browser | O | O | O | O | O |
| Continious Filter | O | X | O | O | O |

**Table 7: The Pattern Developers Categorization, effectivity**

| Pattern | Time to Learn | Speed of Performance | Rate of Errors by Users | Retention over Time | Subjective Satisfaction |
|---|---|---|---|---|---|
| Narrative | O | O | O | O | X |
| High-density Information Display | O | X | O | O | X |
| Status Display | O | O | O | O | X |
| Form | X | X | X | O | O |
| Control Panel, Command Area | O | X | O | X | O |
| WYSIWYG Editor | X | O | X | X | X |
| Composed Command | O | X | O | O | X |
| Navigable Spaces | X | O | O | X | X |
| Overview Beside Detail | O | O | X | O | X |
| Step-by-Step Instructions | X | O | X | O | O |
| Small Groups of Related Things | X | O | O | X | O |
| Hierarchical Set | X | O | O | X | X |
| Tabular Set | O | X | O | O | O |
| Chart or Graph | O | X | O | O | O |
| Optional Detail On Demand | O | O | O | O | O |
| Disabled Irrelevant Things | X | O | X | X | O |
| Pointer Shows Affordance, Mode Cursor | O | O | X | O | O |
| Short Description | X | X | X | O | O |
| Sovereign Posture | O | X | O | X | O |
| Helper Posture | X | X | O | O | O |
| Background Posture | X | X | O | X | O |
| Central Working Surface | X | X | O | X | X |
| Tiled Working Surfaces | O | X | O | O | X |
| Stack of Working Surfaces | O | X | O | O | O |
| Pile of Working Surfaces | O | X | O | O | X |
| Map of Navigable Spaces | X | O | X | X | O |
| Clear Entry Points | X | O | O | X | X |
| Color-Coded Sections | X | X | X | X | X |
| Go Back One Step | O | X | O | O | X |
| Go Back to a Safe Place | O | O | O | O | X |
| Convenient Environment Actions | X | X | X | X | O |
| Localized Object Actions | X | X | X | X | O |
| Actions for Multiple Objects | O | X | O | O | X |
| Choice from a Small Set | O | X | O | O | O |
| Choice from a Large Set | O | X | O | O | O |
| Sliding Scale | X | O | O | X | X |
| Editable Collection | X | O | O | O | O |
| Forgiving Text Entry | O | X | X | O | X |
| Structured Text Entry | X | O | X | O | O |
| Toolbox | X | X | O | X | O |
| User Preferences, Preferences | O | O | O | X | X |
| Personal Object Space | O | X | O | X | X |
| Scripted Action Sequence | O | X | X | O | X |
| User's Annotations | O | O | O | X | O |
| Bookmarks, Favourites | O | X | X | O | X |
| Repeated Framework | X | X | X | X | X |
| Good Defaults | O | X | X | O | X |
| Remembered State | O | X | X | O | X |

| Pattern | Time to Learn | Speed of Performance | Rate of Errors by Users | Retention over Time | Subjective Satisfaction |
|---|---|---|---|---|---|
| Interaction History | O | X | X | O | X |
| Progress Indicator, Progress | O | O | O | O | X |
| Important Message | O | O | X | O | X |
| Reality Check, Shield, Warning | O | O | X | X | O |
| Demonstration | X | O | X | X | O |
| Wizard | O | O | X | O | X |
| Grid Layout | X | X | O | O | O |
| Contextual Menu | X | O | O | O | O |
| Focus | X | O | O | O | X |
| Unambigious Format | X | O | O | O | O |
| Navigating between Spaces | X | O | O | X | X |
| Like in the real world? | O | O | O | X | X |
| Preview | O | X | X | O | X |
| Container Navigation | O | X | O | X | X |
| Setting Attributes | O | X | O | O | O |
| Hinting | X | O | O | O | X |
| List Browser | O | X | O | O | X |
| Continious Filter | O | X | O | O | X |

**Table 8: The Pattern Users Categorization, experience**

| Pattern | Novice or first-time Users | Knowledgeable Intermittent Users | Expert Frequent Users |
|---|---|---|---|
| Narrative | X | X | O |
| High-density Information Display | O | X | X |
| Status Display | X | X | X |
| Form | X | X | X |
| Control Panel, Command Area | X | X | X |
| WYSIWYG Editor | X | X | O |
| Composed Command | O | O | X |
| Navigable Spaces | X | X | X |
| Overview Beside Detail | X | X | X |
| Step-by-Step Instructions | X | X | O |
| Small Groups of Related Things | X | X | X |
| Hierarchical Set | X | X | X |
| Tabular Set | X | X | X |
| Chart or Graph | X | X | X |
| Optional Detail On Demand | X | X | X |
| Disabled Irrelevant Things | X | X | X |
| Pointer Shows Affordance, Mode Cursor | X | X | X |
| Short Description | X | X | O |
| Sovereign Posture | X | X | X |
| Helper Posture | X | X | X |
| Background Posture | X | X | X |
| Central Working Surface | X | X | X |
| Tiled Working Surfaces | X | X | O |
| Stack of Working Surfaces | O | X | X |
| Pile of Working Surfaces | O | X | X |
| Map of Navigable Spaces | X | X | X |
| Clear Entry Points | X | X | X |
| Color-Coded Sections | X | X | X |
| Go Back One Step | X | X | X |
| Go Back to a Safe Place | X | X | O |
| Convenient Environment Actions | X | X | X |
| Localized Object Actions | X | X | X |
| Actions for Multiple Objects | X | X | X |
| Choice from a Small Set | X | X | X |
| Choice from a Large Set | X | X | X |
| Sliding Scale | X | X | O |
| Editable Collection | X | X | X |
| Forgiving Text Entry | X | X | O |
| Structured Text Entry | X | X | O |
| Toolbox | X | X | X |
| User Preferences, Preferences | X | X | X |
| Personal Object Space | O | X | X |
| Scripted Action Sequence | O | X | X |
| User's Annotations | O | X | X |
| Bookmarks, Favourites | X | X | X |
| Repeated Framework | X | X | X |
| Good Defaults | X | X | O |
| Remembered State | X | X | X |
| Interaction History | X | X | X |
| Progress Indicator, Progress | X | X | X |
| Important Message | X | X | X |
| Reality Check, Shield, Warning | X | X | O |
| Demonstration | X | X | O |
| Wizard | X | X | O |

| Pattern | Novice or first-time Users | Knowledgeable Intermittent Users | Expert Frequent Users |
|---|---|---|---|
| Grid Layout | X | X | X |
| Contextual Menu | X | X | O |
| Focus | X | X | X |
| Unambigious Format | X | X | X |
| Navigating between Spaces | X | X | X |
| Like in the real world? | X | X | X |
| Preview | X | X | X |
| Container Navigation | X | X | X |
| Setting Attributes | X | X | X |
| Hinting | X | X | O |
| List Browser | X | X | X |
| Continious Filter | X | X | X |

**Table 9: The Pattern Users Categorization, interaction style**

| Pattern | Direct Manipulation | Menu Selection | Form Fillin | Command Language | Natural Language |
|---|---|---|---|---|---|
| Narrative | X | X | X | O | X |
| High-density Information Display | X | O | O | O | O |
| Status Display | O | O | O | O | O |
| Form | O | O | X | O | O |
| Control Panel, Command Area | X | O | O | O | O |
| WYSIWYG Editor | X | O | O | O | O |
| Composed Command | O | O | O | X | O |
| Navigable Spaces | X | O | O | O | O |
| Overview Beside Detail | O | O | O | O | O |
| Step-by-Step Instructions | X | O | X | O | O |
| Small Groups of Related Things | X | X | X | O | O |
| Hierarchical Set | X | X | O | O | O |
| Tabular Set | X | O | X | O | O |
| Chart or Graph | X | O | O | O | O |
| Optional Detail On Demand | X | O | O | O | O |
| Disabled Irrelevant Things | X | O | O | O | O |
| Pointer Shows Affordance, Mode Cursor | X | O | O | O | O |
| Short Description | X | O | O | O | O |
| Sovereign Posture | X | X | X | X | X |
| Helper Posture | X | X | X | X | X |
| Background Posture | X | X | X | X | X |
| Central Working Surface | X | O | O | O | O |
| Tiled Working Surfaces | X | O | O | O | O |
| Stack of Working Surfaces | X | O | O | O | O |
| Pile of Working Surfaces | X | O | O | O | O |
| Map of Navigable Spaces | X | O | O | O | O |
| Clear Entry Points | X | O | O | O | O |
| Color-Coded Sections | X | O | O | O | O |
| Go Back One Step | X | O | O | O | O |
| Go Back to a Safe Place | X | O | O | O | O |
| Convenient Environment Actions | X | O | O | O | O |
| Localized Object Actions | X | O | O | O | O |
| Actions for Multiple Objects | X | O | O | O | O |
| Choice from a Small Set | X | X | O | O | O |
| Choice from a Large Set | X | X | O | O | O |
| Sliding Scale | O | O | O | O | O |
| Editable Collection | X | O | O | O | O |
| Forgiving Text Entry | O | O | O | X | X |
| Structured Text Entry | O | O | X | O | O |
| Toolbox | X | O | O | O | O |
| User Preferences, Preferences | X | O | O | O | O |
| Personal Object Space | X | O | O | O | O |
| Scripted Action Sequence | X | O | O | X | O |
| User's Annotations | X | X | X | O | O |
| Bookmarks, Favourites | X | O | O | O | O |
| Repeated Framework | X | O | O | O | O |
| Good Defaults | X | O | X | O | O |
| Remembered State | X | O | X | O | O |
| Interaction History | X | O | O | X | X |
| Progress Indicator, Progress | X | O | O | X | X |
| Important Message | X | X | X | X | X |
| Reality Check, Shield, Warning | X | X | X | X | X |
| Demonstration | X | O | O | O | O |

| Pattern | Direct Manipulation | Menu Selection | Form Fillin | Command Language | Natural Language |
|---|---|---|---|---|---|
| Wizard | X | O | O | O | O |
| Grid Layout | X | O | X | O | O |
| Contextual Menu | O | X | O | O | O |
| Focus | X | O | O | O | O |
| Unambigious Format | O | O | X | O | O |
| Navigating between Spaces | X | O | O | O | O |
| Like in the real world? | X | O | O | O | O |
| Preview | O | O | O | O | O |
| Container Navigation | X | O | O | O | O |
| Setting Attributes | X | O | O | O | O |
| Hinting | X | O | O | O | O |
| List Browser | X | O | O | O | O |
| Continious Filter | X | O | O | O | O |

**Table 10: The Pattern Users Categorization, interface level**

| Pattern | Application | Containers | Widgets |
|---|---|---|---|
| Narrative | O | O | X |
| High-density Information Display | O | X | O |
| Status Display | O | O | X |
| Form | O | X | O |
| Control Panel, Command Area | O | X | O |
| WYSIWYG Editor | X | O | O |
| Composed Command | X | O | O |
| Navigable Spaces | X | O | O |
| Overview Beside Detail | O | X | O |
| Step-by-Step Instructions | X | O | O |
| Small Groups of Related Things | O | X | O |
| Hierarchical Set | O | X | O |
| Tabular Set | O | X | O |
| Chart or Graph | O | X | O |
| Optional Detail On Demand | O | X | O |
| Disabled Irrelevant Things | O | X | O |
| Pointer Shows Affordance, Mode Cursor | O | O | X |
| Short Description | O | O | X |
| Sovereign Posture | X | O | O |
| Helper Posture | X | O | O |
| Background Posture | X | O | O |
| Central Working Surface | X | O | O |
| Tiled Working Surfaces | X | O | O |
| Stack of Working Surfaces | X | O | O |
| Pile of Working Surfaces | X | O | O |
| Map of Navigable Spaces | X | O | O |
| Clear Entry Points | X | O | O |
| Color-Coded Sections | O | X | O |
| Go Back One Step | X | O | O |
| Go Back to a Safe Place | X | O | O |
| Convenient Environment Actions | X | O | O |
| Localized Object Actions | X | O | O |
| Actions for Multiple Objects | O | X | O |
| Choice from a Small Set | O | O | X |
| Choice from a Large Set | O | O | X |
| Sliding Scale | O | O | X |
| Editable Collection | O | O | X |
| Forgiving Text Entry | O | O | X |
| Structured Text Entry | O | O | X |
| Toolbox | O | X | O |
| User Preferences, Preferences | X | O | O |
| Personal Object Space | X | O | O |
| Scripted Action Sequence | X | O | O |
| User's Annotations | X | O | O |
| Bookmarks, Favourites | X | O | O |
| Repeated Framework | X | O | O |
| Good Defaults | O | O | X |
| Remembered State | O | O | X |
| Interaction History | X | O | O |
| Progress Indicator, Progress | O | O | X |
| Important Message | X | O | O |
| Reality Check, Shield, Warning | X | O | O |
| Demonstration | X | O | O |
| Wizard | X | O | O |
| Grid Layout | O | X | O |

| Pattern | Application | Containers | Widgets |
|---|---|---|---|
| Contextual Menu | X | O | O |
| Focus | O | O | X |
| Unambigious Format | O | O | X |
| Navigating between Spaces | X | O | O |
| Like in the real world? | O | O | X |
| Preview | O | O | X |
| Container Navigation | O | X | O |
| Setting Attributes | O | X | O |
| Hinting | O | O | X |
| List Browser | X | O | O |
| Continious Filter | O | O | X |