INFORMATION SOCIETIES TECHNOLOGY (IST) PROGRAMME

STATUS

"Software Architecture for Usability"

WORKPACKAGE: 2 – Usability Attributes Affected by Software Architecture

D2 : Usability Attributes Affected by Software Architecture

Version: 1.1

Submission Date: 01/06/2002

Authors: Alberto de Andres, Jan Bosch, Avratoglou Charalampos, Robert Chatley, Xavier Ferre, Eelke Folmer, Natalia Juristo, Jeff Magee, Stavros Menegos, Ana Moreno

Partners: ICSTM, RUG, UPM, IHG, LogicDIS

Stage:	Confidentiality:
[ ] Draft	[ ] Public - for public use
[ ] To be reviewed by WP participants	[ ] IST – for IST programme participants
[ ] Pending of approval by next consortium meeting	[x] Restricted – for STATUS consortium and PO
[x] Final / Released to CEC	

DOCUMENT CONTROL

Registration of Changes

Date	Version	Author of Changes	Comments
19/4/02	0.1	ICTSM	Preliminary version
24/4/02	0.2	ICTSM	Comments from partners
29/4/02	1.0	ICTSM	Final adjustments of format and comments from UPM
30/4/02	1.1	UPM	Adjustments on cover page

List of STATUS Related Documents

Document Name	Version
Technical Annex	
D.1.1. Periodic Progress Report	1.0

1. INTRODUCTION AND OVERVIEW

This deliverable reports on the work carried out in Workpackage 2 on Usability Attributes affected by Software Architecture. As conceived in the Technical Annex, it was envisaged that this workpackage would look at a decomposition of the basic Usability Attributes into more detailed attributes based on a survey of the literature and from the experience of the industrial partners. The idea was then to see which of these detailed attributes affected design decisions at the architectural level. Initial attempts at pursuing this approach showed that a neat partition of Usability Attributes into those that affect architectural decisions and those that do not, did not exist. More critically, the decomposed usability attributes did not map well to the usability issues that were found to be of importance for the industrial partners. For example, both industrial partners identified consistency of interface and operation to be critical to the usability of their systems. However, while this consistency clearly relates to the classic usability attribute of *Learnability*, it is not obviously a direct subcategory or subclass of Learnability Consequently, a more indirect approach has been taken in establishing the relationship between Usability Attributes and Software Architecture. In essence, we have identified issues such as consistency to be *Usability Properties* and looked at how these contribute to the usability of a system. Further, the report looks at how system level solutions that we have termed *Usability Patterns* contribute to the Usability Properties that hold for a system. The approach is outlined in this introduction and substantiated in detail in the following sections.

1.1 Usability and Software Architecture

The ultimate evaluation of quality is fitness for purpose. In order to measure software quality we have to understand the purpose for which the system is intended. Thus quality is not a measure of software in isolation; it is a measure of the relationship between software and its application domain. The user is an essential part of such a domain, so usability is an important component of software quality. Although there is no agreed set of critical software quality attributes, several quality attribute classifications agree on the importance of considering usability as a quality attribute [IEEE1061, 98], [ISO9126, 91], [Boehm, 78]. Most quality attributes focus on characteristics that are desirable from the point of view of the software development organisation. Usability, on the other side, is a quality attribute perceived directly by the client/user. One definition of usability is quality in use [ISO14598, 99].

The relationship between usability and the other quality aspects such as reliability, performance and security is a complex one. Usability can sometimes conflict with other quality attributes. For example, security is the attribute that most evidently conflicts with usability, as security procedures often get in the way of the user's task. Other attributes, such as reliability, often have a positive effect on usability: a software product that is not reliable can hardly support the user in his/her duty, and for this reason it cannot be seen as having a high level of usability. However, the reverse is not true. A software system that is highly reliable could be quite unusable. Defining the borders between usability and the other quality aspects (mainly functionality) is thus a difficult task.

Fundamentally, usability reflects how easy the system is to learn and use, how productively users will be able to work and how much support users need. A system's usability deals not only with the user interface, it also relates closely to the software's overall structure and to the concept on which the system is based. In order to understand the depth and scope of the usability of a system it is useful to make a distinction between the visible part of the user interface (buttons, pull-down menus, check-boxes, background colour, etc.) and the interaction part of the system. By interaction we mean the coordination of information exchange between the user and the system.

This interaction must be carefully designed and must be considered not just when designing the visible part of the user interface, but also when designing the rest of the system. For example, if we consider that our system will have to provide continuous feedback to the user for usability reasons, we will need to bear this in mind when designing time-consuming system operations. These have to be designed so as to allow information to be frequently sent to the user interface to keep the user informed

about the current status of the operation. This information could be displayed to the user by means of a percentage-completed bar, as in some software installation programs. It is not unusual to find development teams thinking that they can design the system and then make it usable afterwards just by designing a nice set of controls, having the right colour combination and the right font. This approach is clearly wrong. The interaction with the user must be considered from the beginning of the development process in order to obtain a usable system. In other words, interaction with the user must be considered when determining the overall architecture of a system.

To define what we mean by the architecture of a software system, we refer to the following definition [Bass, 97]:

"The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them."

Bass, Clements and Kazman go on to explain that: "By "externally visible" properties, we are referring to those assumptions other components can make of a component, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on. The intent of this definition is that a software architecture must abstract away some information from the system (otherwise there is no point looking at the architecture, we are simply viewing the entire system) and yet provide enough information to be a basis for analysis, decision making, and hence risk reduction." Our intention in the STATUS project is to provide a sound basis for analysis and design at this architectural level with respect to usability.

1.2 Usability Attributes, Properties and Patterns

Usability Attributes

Section 2 of this report presents a comprehensive survey of what different researchers in the field of Usability mean by a Usability Attribute. It emerges that the generally accepted meaning is that a *Usability Attribute is a precise and measurable component of the abstract concept that is usability.* In Section 2, we identify the following set of Usability Attributes to be considered by STATUS. In summary, these are:

- **Learnability** – how quickly and easily users can begin to do productive work with a system that is new to them, combined with the ease of remembering the way a system must be operated.
- **Efficiency of use** – the number of tasks per unit time that the user can perform using the system.
- **Reliability** – sometimes called "reliability in use", this refers to the error rate in using the system and the time it takes to recover from errors.
- **Satisfaction** – the subjective opinions that users form in using the system.

These attributes can be measured directly by observing and interviewing users of the final system using techniques that are standard in the area of usability engineering. However, as we discuss at the beginning of this section, decomposition of these attributes to more detailed elements does not lead to a convincing connecting relationship with architecture – why? The answer would appear to be that these are attributes that we can measure from the system rather than being direct requirements that can be refined into architectural design decisions. For example, we could say that a goal for a system is that it should be easy to learn, or that new users should require no more than 30 minutes instruction, however, a requirement at this level does not help guide the design process. We need usability requirements to take a more concrete form expressed in terms of the solution domain to influence architectural design.

Usability Properties

We refer to those usability requirements of a system that are more directly related to the solution domain and which have a direct relationship to software design decisions as Usability Properties. Essentially, these properties embody the heuristics and design principles that researchers in usability have found to have a direct influence on system usability. Section 3 of this report has a comprehensive survey of this work and arrives at an initial classification for use by STATUS. In summary, we classify usability properties into:

- **Providing feedback** – the system provides continuous feedback as to system operation to the user.
- **Error Management** – includes error prevention and recovery.
- **Consistency** – consistency of both the user interface and functional operation of the system.
- **Guidance** – on-line guidance as to the operation of the system.
- **Minimize cognitive load** – system design should recognize human cognitive limitations, short-term memory etc.
- **Natural Mapping** – includes predictability of operation, semiotic significance of symbols and ease of navigation.
- **Accessibility** – includes multi-mode access, internationalisation and support for the disabled.

We noted above, these Usability properties captured more directly the usability concerns of the industrial partners, they can be accommodated at design time and have an influence on architectural decisions. Fore example, the requirement to provide feedback may require that additional communication paths and interaction interfaces are included in the description of the system's software architecture. However, these usability properties are still far removed from the standard mechanisms and techniques that are often found in systems to aid usability. It is the case that the requirement for a usability property such as guidance is usually met by the provision of a combination of highlighting active buttons, wizards and context sensitive help. How do we capture this relationship between commonly occurring usability requirements or properties and the solutions that can be adopted at design time?

Usability Patterns

Borrowing from the world of design patterns in which a design pattern is a reusable solution to a commonly occurring problem, we have coined the term *Usability Pattern* to refer to a technique or mechanism that can be applied to the design of the architecture of a system in order to address a need identified by a usability property. In section 4, we have collected 20 such usability patterns. Each pattern has:

- **Name** – indicative of its purpose
- **Description** – a brief description of the mechanism or technique
- **Relationship with Software Architecture** – outlines the architectural impact of the pattern.
- **Relationship with Usability Properties** – identifies the usability properties the pattern relates to.
- **Example** – an example of the use of the pattern in a system that is in use.

The usability patterns collected in section 4 represent the initial work in STATUS to relate architecture to usability. It is envisage that the collection will be both expanded in coverage and level of detail as the project progresses.

Relating attributes, properties and patterns:

In clarifying the relationship between measurable usability attributes, usability properties and usability patterns, the project members have found the following diagram useful. It outlines an iterative design process with respect to usability and architecture.

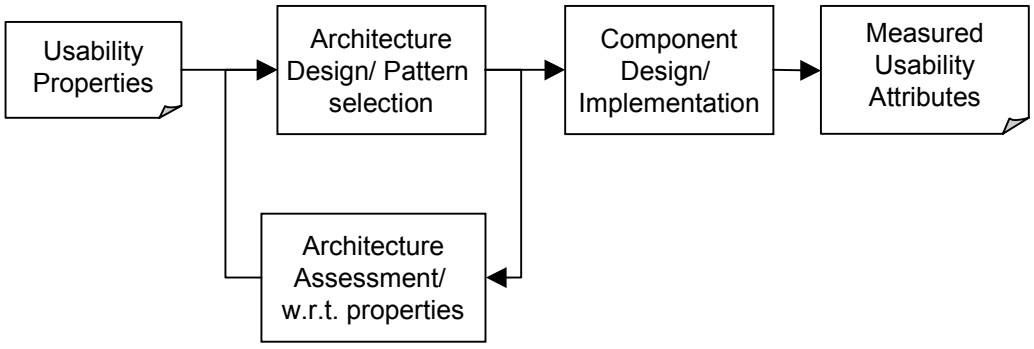


Figure 1 – Design Process and Usability

In essence, the usability properties form part of the requirements that drive the process of architectural design. This is achieved partly by selecting appropriate design patterns to satisfy the required properties. We can assess the architecture with respect to these properties and this assessment will inform redesign. After components have been designed and implemented, the resulting system can be tested/measured with respect to its Usability Attributes. Clearly, if usability is unsatisfactory at this stage, both component and architecture redesign may be necessary. An objective of STATUS is to minimise the redesign necessary at this late stage and as far as is possible ensure satisfactory usability at the architectural design stage. To do this, we need to establish the relationship between usability properties and usability attributes. The eventual goal is a predictive model that will allow architectural assessment to predict the final usability attributes that can be achieved by the implemented system. As an initial step, we have identified the qualitative relationships between properties, attributes and patterns. These relationships are captured in the table presented in Section 5, a fragment of which is shown below in Figure 2.

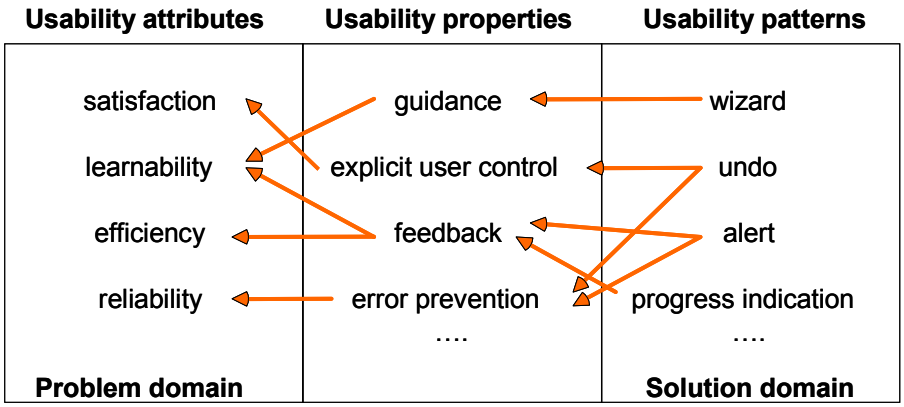


Figure 2 – Attribute, Property & Pattern relationships

For example, in figure 2 the wizard pattern improves learnability, but how? The wizard pattern uses the concept of *guidance* to take the user through a complex task one step at a time. Guidance in its turn improves the learnability usability attribute. The concept of *guidance* is a usability property. Patterns address one or more of the usability properties. Usability properties relate patterns to usability attributes in the figure in the qualitative sense that an arrow indicates that a property positively affects an attribute (i.e. improves that attribute). The usability properties that a pattern relates to are documented in each pattern.

In the following, section 2 surveys the literature in usability attributes and justifies the usability attributes chosen for use in STATUS, section 3 surveys the various usability heuristics, design guidelines and ergonomic principles that lead to the usability property classification we have adopted, section 4 defines what is meant by a usability pattern and presents our initial collection of patterns, and finally, section 5 presents a table summarising the relationships between attributes, properties and patterns that we have so far identified.

2. USABILITY ATTRIBUTES

Usability is an issue we can approach from multiple angles. Many different disciplines, such as psychology, computer science, and sociology, are trying to tackle it. Unfortunately, this results in a lack of standards; a lot of terms are used to describe the development of usable software: usability engineering, usage-centered design, contextual design, participatory design, and goal-directed design. All these philosophies to some extent address the core issue of evaluating usability with real users from the first stages of development, and keeping a user-centered focus throughout the development effort.

A paradigm more widely accepted in software engineering in relation to usability is Usability Engineering. Usability engineering defines a target usability level in advance and ensures that the software developed reaches that level. The term was coined to reflect the engineering approach some usability specialists propose to take. Specifically, usability engineering is defined as "a process through which usability characteristics are specified, quantitatively and early in the development process, and measured throughout the process" [Hix, 93].

For a better understanding of the concept of usability, it is necessary to consider its attribute decomposition, which reflects the different aspects that must be taken into account. This section presents a survey of attribute decompositions found in the literature and concludes with the set of usability attributes to be used in the STATUS project.

2.1 Sources for the Literature Survey

The issue of usability attribute decomposition is addressed by different authors in the usability literature. For this review we will just focus on books since we want our starting point to be well established knowledge in the field, not the novel and not-yet-accepted ideas that might be published in research papers.

From among the many usability and human computer interaction books we have chosen the ones most relevant and most often cited for the usability attributes survey. A brief description of the six chosen books follows.

- Nielsen93 – "Usability Engineering". Jakob Nielsen.

  This book has been the main reference book for the discipline of usability engineering for a long time. Nielsen offers an engineering-like approach to building usable software systems, which brings usability issues closer to a Software Engineering view.

- Hix93 – "Developing User Interfaces: Ensuring Usability Through Product and Process" D. Hix and H. Hartson.

  This book presents a very practical and hands-on approach to the issue of user interaction design. One of its objectives is to be a textbook for courses in user interface development with a strong usability focus.

- Wixon97 – "The Usability Engineering Framework for Product Design and Evaluation" D. Wixon and C. Wilson. In *Handbook of Human-Computer Interaction, 2nd edition*. The Handbook contains articles that describe a diverse set of topics in HCI, both in research and practice. Wixon and Wilson's article gives a good overview of Usability Engineering. The authors belonged to the usability group at DEC that created the method Usability Engineering (as credited in [Gould, 88]).

- Constantine99 – "Software for Use". Larry L. Constantine, Lucy A.D. Lockwood. Larry Constantine is one of the gurus of the Software Engineering discipline. He has shifted in the last decade to concentrate on the issue of developing usable software. His and

Lockwood's experience as usability consultants is described in this very practical work. The book presents the authors' own method for developing usable software.

- Shackel91 – "Usability – context, framework, design and evaluation". B. Shackel. In *Human Factors for Informatics Usability*.

  Shackel is one of the first authors in the field to recognise the importance of usability engineering and the relevance of the concept of usability. His approach has been much used and modified.

- Preece94 – "Human-Computer Interaction". J. Preece, , Y. Rogers, H. Sharp, D. Benyon, S. Holland, T. Carey.

  This book presents a wide variety of topics addressed by the HCI field. It has an encyclopaedic aim, with some theoretical prevalence. It has been one of the main textbooks for general HCI courses to the time of writing.

- Shneiderman98 – "Designing the User Interface". Ben Shneiderman. Shneiderman is one of the most respected authors in the HCI field (last year he received the ACM-SIGCHI CHI Lifetime Achievement Award). All three editions of this book have been fundamental references in user interface design, because of their balanced coverage of both theoretical and development-oriented aspects of interaction.

- ISO 9126_00[1] – "ISO 9126-1 Software engineering – Product Quality – part 1: Quality Model". International Organization for Standardization.

  ISO9126 defines a quality model for Software Engineering. It is relevant for our study, because it includes usability among the six categories of software quality that are relevant during product development (functionality, reliability, usability, efficiency, maintainability and portability).

2.2 Definition of Usability Attribute

What do we understand the term "usability attribute" to mean? Usability is an abstract concept that needs to be decomposed into measurable components; these components of usability are what we will refer to as attributes. Not everybody calls these components "usability attributes": for the shake of clarity, table 1 shows the terms used by the authors we consider.

Constantine99	Hix93	ISO9126_00	Nielsen93	Preece94	Shackel91	Shneiderman98	Wixon97
Usability facets	Usability attributes	Usability attributes	Usability attributes	Components of usability	Scales of usability	Measurable human factors	Usability attributes

Table 1 – Terms used for the concept of Usability Attribute

The definition given by each author for the term is as follows:

- Usability facets are different aspects of a system and its user interface that contribute to usability [Constantine, 99].
- The usability attribute is the general usability characteristic to be measured for an interface [Hix, 93].

[1] ISO standards definitions are taken from [Bevan, 01]

- Usability attributes: a set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users [ISO 9126,00] try to identify those attributes that influence "the effort needed for use" [ISO 9126,00].
- Nielsen does not give an explicit definition for usability attributes, but they are characterized as being precise and measurable components of the abstract concept of usability [Nielsen, 93].
- Components of usability. The authors don't give an explicit definition, but they describe them as being operationalized so that they can be tested [Preece, 94].
- Usability Scales. Shackel suggests a set of operational criteria; for a system to be usable; it has to achieve defined levels on a set of scales [Shackel, 91].
- Measurable human factors: Precise measurable objectives that guide the designer, evaluator, purchaser or manager [Shneiderman, 98].
- Usability attributes are characteristics of a product that can be measured in some quantitative way [Wixon, 97].

As can be seen, most of the definitions focus on measurable attributes oriented to usability evaluation purposes.

2.3 Usability Attributes Proposals

For each source considered we have extracted its usability attribute decomposition, to include it as a column in Table 2. We wanted to be able to compare easily the different decompositions, so we have grouped attributes that refer to the same concept in the same row. We have chosen to name each row (first column in the table) with the most general term, or the one that appears most amongst the different authors.

Where the author offers a second name for the same concept, it is detailed between brackets, for example 'Efficiency (long-term performance)' . For attributes not mentioned in the source, the cell contains a dash ('-').

The first five attributes are the most mentioned by all authors in the field, so we have separated them by means of a double bar from the other five attributes identified.

2.4 Attributes Description

Different authors give different definitions for each attribute. In order to reach a consensus on each attribute's meaning, we will review in the following sections each author's definition, along with our own definition where necessary.

2.4.1 Learnability

Learnability has to do with how quickly and easily users can begin to do productive work with a system that is new to them. Ease of learning is another wording for this usability attribute, as it is mentioned by most sources ([Constantine, 99] [Hix, 93] [Nielsen, 93] [Shneiderman, 98]). [Shackel, 91] gives a slightly different definition for learnability: degree of learning to accomplish tasks.

Learnability is the most important attribute for novice users, as it defines how much time they will spend being novices before being considered as proficient in the use of the system (or at least expert enough so they can easily use the system) [Nielsen, 93][Hix, 93] [Preece, 94].

This attribute is linked to the speed of a user's evolution from being a novice in his/her usage of the system to being an expert (or expert enough that a certain level of performance can be reached). This evolution can be accelerated by cutting down the set of features offered to the novice user, and then gradually revealing the full complexity of the system to the user as his/her knowledge of the system

increases. A user interface based on user modes (novice, normal and advanced user modes, for example) could help to achieve a good level of learnability.

Attribute\Source	**Constantine99**	**Hix93**	**ISO 9126_00**	**Nielsen93**	**Preece94**	**Shackel91**	**Shneiderman98**	**Wixon97**
Learnability (time to learn)	Learnability	Learnability	Learnability	Learnability (ease of learning)	Learnability (ease of learning)	Learnability (time to learn)	Time to learn	Learnability (initial performance)
Efficiency in use	Efficiency in use	Long-term performance	Operability (?)	Efficiency of use	Throughput	Effectiveness (speed)	Speed of performance	Efficiency (long-term performance)
Learnability (retention over time)	Rememberability	Retainability	--	Memorability	-	Learnability (retention)	Retention over time	Memorability
Reliability in use	Reliability in use	-	Operability (?)	Errors	Throughput	Effectiveness (errors)	Rate of errors by users	Error rates
Likability (long-term satisfaction)	User satisfaction	Long-term user satisfaction	Attractiveness	Satisfaction	Attitude	Attitude	Subjective Satisfaction	Satisfaction or likability
Understandability			Understandability	-	-	-	-	-
Adaptability	-	-	-	-	Flexibility	Flexibility	-	Flexibility
First impression	-	First impression	-	-	-	-	-	First impressions
Extra features	-	Advanced feature usage	-	-	-	-	-	Advanced feature usage
Initial efficiency	-	Initial performance	-	-	-	-	-	Learnability (initial performance)
Evolvability	-	-	-	-	-	-	-	Evolvability

Table 2 – Decomposition Proposals for Usability Attributes

Despite being one of the attributes with most similarity amongst different authors' definitions, it is interesting to note that even in this attribute there are discrepancies. For example, [Wixon, 97] mentions learnability as being equivalent to initial performance (see discussion on this issue in 2.4.10 below).

To summarise, we will consider learnability as being how quickly and easily users can reach a level of proficiency in using the system.

2.4.2 Efficiency

According to Nielsen [Nielsen, 93], efficiency refers to an expert user's steady-state level of performance at the point where the learning curve flattens out. Nielsen relates this attribute to progress along the learning curve, establishing a link with the concept of learnability.

[Hix, 93] and [Wixon, 97] specify efficiency as long-term performance, therefore associating it to an extent with expert usage as well.

[Shackel, 91] describes effectiveness as performance in accomplishment of tasks, considering both speed and errors.

Efficiency is described by [Constantine, 99] as the level of user productivity while using the system. These authors remark that this attribute does not relate to the efficiency of the computer on its own, but to the efficiency of the user-computer combination.

[Shneiderman, 98] restricts its application to a set of benchmark tasks.

Not clearly in this category, but somehow related, is the operability attribute. It is defined in [ISO9126, 00] as the capability of the software product to enable the user to operate and control it.

We can summarise efficiency as the number of tasks per unit of time that the user can perform using the system.

2.4.3 Memorability

Retention over time, rememberability or memorability refers to the ease of remembering the way a system must be operated.

[Nielsen, 93] describes this as the characteristic of a system that allows the user to return to the system after some period of not having used it, without having to learn everything all over again. [Hix, 93] [Shneiderman, 98] and [Wixon, 97] refer as well to a period of non-usage for the definition of this attribute.

[Shneiderman, 98] closely links retention over time and learnability. We agree on this strong relationship between both attributes, that can be of interest for the definition of an attribute hierarchy. In this same direction, Shackel understands learnability as covering both time to learn and retention over time [Shackel, 91].

2.4.4 Error rate

Error rate refers to the errors made during the use of the system and how easy it is to recover from them [Nielsen, 93].

A system should have few and non-catastrophic errors, so good error handling is critical for the usability of a system [Shneiderman, 98].

According to Shneiderman and Nielsen, errors can have an impact on efficiency, by slowing down performance. [Preece, 94] goes in the same direction by defining a unique attribute named throughput, which considers together the accomplishment of tasks, the speed of task execution and the errors made. This kind of attribute grouping is of interest for the definition of an attribute hierarchy. A similar approach is taken by Shackel, connecting speed of performance to errors in his attribute called effectiveness [Shackel, 91],

[Constantine, 99] uses a term which is very commonly found between the aspects of software quality handled by Software Engineers: reliability. But the system considered by Constantine and Lockwood (as in section 2.4.2 above) is not just the hardware-software but also includes the user. So these authors use the term "Reliability in use", taking into account the error-prone nature of human users. Errors that were previously blamed on the user can be now regarded as possible consequences of usability failings. This way of considering usability issues is shared by other authors in the field, who prefer the term 'quality of use' [Bevan, 95].

Easy recovery from errors should be pursued by any software system [Nielsen, 93]. This kind of design principle can be a starting point for architectural decisions looking for an improvement in this usability attribute.

2.4.5 Satisfaction

Satisfaction is the subjective opinion that users form about the system (or about some parts of it).

Hix and Hartson distinguish between the first impression (see section 2.4.8 below) and long-term user satisfaction,. The latter refers to the opinion of the user after using the system for a longer period of time [Hix, 93]. This kind of distinction can be of use in the attribute hierarchy definition.

[Constantine, 99] claims that satisfying software is likely to be used more often and used more effectively, therefore relating this attribute to efficiency as a core usability attribute.

On the other hand, [ISO9126,00] uses attractiveness: "the capability of the software product to be attractive to the user".

Satisfaction is the most elusive usability attribute, as it is completely dependant on subjective opinion of users. For this reason, it seems to be the usability attribute less interesting a priori for the objective of the STATUS project.

2.4.6 Understandability

[ISO9126,00] standard states that understandability is the capability of a software product to enable the user to understand whether the software is suitable, and how it can be used, for particular tasks and conditions of use. This attribute has more to do with provision of the right functionality; e.g. matching the software to the user's needs.

2.4.7 Flexibility

[Shackel, 91] defines this attribute as adaptation to variation in tasks. [Preece, 94] extends this definition by including changes in the environment; therefore, it defines flexibility as the extent to which the system can accommodate changes to the tasks and environments beyond those first specified. When the usability of a software system reaches an upper level, users can spontaneously begin to find new uses for the system. But this is rare and this characteristic cannot be easily tackled when developing the system.

[Wixon, 97] defines it as the extent to which the product can be applied to different kinds of tasks.

All three definitions are similar, but they can be understood diversely, so we will not try to extract a common definition.

2.4.8 First impression

First impression is a usability attribute that specifies the user's opinion on the system after being using it for a short period of time [Hix, 93]. It is an attribute as elusive as satisfaction for our purpose.

Nielsen mentions an *approachability* attribute that is the impression that gets the user about the system ease of use without having actually used it [Nielsen, 93]. This attribute is not exactly first impression,

but it is closely related. It is the impression that the potential buyer gets even before having used the system for the first time.

2.4.9 Advanced feature usage

A proper definition is not given in [Hix, 93] (the only considered source that mentions it). It is presented as an attribute that helps determine usability of more complicated functions of an interface.

It can be related with efficiency because its relationship with expert users.

2.4.10 Initial performance

Initial performance is a user's performance during the very first use of an interface by a user [Hix, 93].

[Wixon, 97] narrows the definition of learnability to being initial performance. Undoubtedly, initial performance is closely related to learnability, but there is more to the ease of learning of a software system than the initial performance of a novice user. Anyway, it is true that initial performance can reveal learnability problems.

2.4.11 Evolvability

Evolvability is a kind of flexibility, but focused on the user. It deals with how well the system adapts to changes in user expertise [Wixon, 97].

The transition from novice to expert users present in the definition of attributes like learnability and efficiency, is also the key concept for the evolvability attribute.

The use of different levels or modes of usage in the user interface can help to get the desired level of evolvability (see section 2.4.1 above).

2.5 Other Proposals

Some usability bibliographical sources deal with the issue of usability attributes in a less formal manner, or they have a different approach. Nevertheless, we consider that their view can be of interest, as it can enrich our vision. These additional proposals are presented in this section.

2.5.1 [Mayhew, 99]

[Mayhew, 99] presents the necessity of establishing measurable usability goals. These goals serve two purposes: To help focus user interface design efforts, and to serve as acceptance criteria during usability evaluation. The definition of usability goals is based in our concept of usability attributes, but the author does not define such kinds of attributes. Instead, she presents several different classifications of usability goals as follows:

- Qualitative use goals (not quantified) vs. Quantitative goals (objective and measurable).
- Ease-of-use goals (focus on the use of the product by experienced users who have been trained on how to use the product) vs. Ease-of-learning goals (focus on the use of the product by first-time users).
- Absolute (absolute quantification) vs. Relative (user's experience on the product relative to previous experiences on some benchmark product).
- Performance goals (quantify actual user performance while using a product to perform a task) vs. Preference/Satisfaction goals (a user preference among alternative interfaces / the level of satisfaction with a particular interface).

2.5.2 [ISO9241_98] [2]

The ISO organisation has developed various usability-related standards over the last 15 years. The one referenced in the above section "ISO 9241-11 Ergonomic Requirements for Office Work with Visual Display Terminals – Part 11: Guidance on Usability" provides the definition of usability that is used in ergonomic standards. ISO standards on ergonomic requirements e.g. VDT workstation, hardware & environment have been widely adopted by industry.

In this standard usability is evaluated not from a consumer product acceptance point of view but from a quality of work point of view. [ISO9241, 98] refers to the Dimensions of usability consisting of:

- user performance view (effectiveness, efficiency)
- user view (satisfaction).

2.5.3 [Rohlfs, 98]

This work presents a particular solution for the problem of redesigning legacy systems. It does not include a detailed definition of usability attributes, but it mentions the targets for the most common usability performance objectives:

- Task-specific error rates.
- Task-specific use of online-help.
- Task-specific and/or overall satisfaction with ease of use or efficiency of use.
- Overall guessability (e.g. 80% of the icons guessed correctly the first time).
- Overall satisfaction with ease of use or efficiency of use.

2.5.4 [Constantine, 99]

Constantine defines five core usability criteria (as they appear in section 2.4). But he also mentions some additional usability criteria which often come into play, without giving much detail on them. They are the following:

- Accuracy.
- Clarity of presentation.
- Comprehensibility.
- Flexibility of operation.
- Ease of navigation.

2.6 Usability Decomposition for the STATUS Project

One of the unresolved questions concerning usability, which is intrinsically related to its decomposition into attributes, is the scope this software quality element is to be given. It is interesting to note that not all of the attributes identified originally in Table 2 are at the same level of abstraction. Attributes like learnability or reliability are attributes according to the traditional definition in software evaluation, and they are not directly observable at the system (usability tests or questionnaires are necessary). On the

[2] ISO standards definitions are taken from [Bevan, 01]

contrary, some others like adaptability are more directly observable, as either the system has support for some related functionality or it has not. For that reason when talking about usability characteristics, a structured organisation of such characteristics is required, as was mentioned in section 1.2, where the reasoning followed to distinguish between usability attributes, properties and patterns was described.

According to that reasoning, in identifying the usability attributes for STATUS project we have chosen to take as the basis the view that appears to be shared by the majority of relevant authors in the field. As shown in Table 2, this view considers that usability can be decomposed into four basic attributes:

- Learnability
- Efficiency of use
- Reliability
- Satisfaction

This view, therefore, perfectly delimits and separates usability from the other software quality elements such us efficiency, functionality, maintainability or portability.

Note also that in the final usability decomposition to be considered in the STATUS project, attrbiutes related to memorability have been considered jointly with learnability, in accordance with the views of authors like Shackel [Shackel, 91].

Another reason to consider the classic decomposition into usability attributes for the STATUS project is that those attributes are the ones usually measured in the final software systems to privide specific levels of usability [Nielsen, 93]. So the intention of the STATUS partners is to provide specific ways to measure those same attributes during the software design.

3. USABILITY PROPERTIES

We refer to those usability requirements of a system that are more directly related to the solution domain and which have a direct relationship to software design decisions as Usability Properties. Essentially, these properties embody the heuristics and design principles that researchers in usability have found to have a direct influence on system usability. Feedback provided by industry parnters about usability requirements from a practical point of view, have also many common points with classical desig heuristics for usability. That's why usability properties also cover industrial point of view about usability. This section is comprehensive survey of this work and arrives at an initial classification of usability properties for use by STATUS.

3.1 Sources for the Literature Survey

The information for this survey comes largely from the same sources that were considered in the previous section when surveying usability attributes.

A brief description of other sources used for this section is below:

- Baecker, 95 – "Readings in Human-Computer Interaction: Toward the year 2000". R.M. Baecher, J. Grudin, W.A.S. Buxton, S. Greenberg.

  This book is a compilation of readings which have been fundamental in the field, along with extended introductions to each issue. In the chapter dedication to Design and Evaluation there is a survey of design principles which is used in this document in sections 3.2.6 to 3.2.8.

- Scapin, 97 - "Ergonomic criteria for evaluating the ergonomic quality of interactive systems, D.L.Scapin, J.M.C. Bastien , Behaviour & Information Technology, vol 16, no 4/5, pp.220-231.
- McKay, 99 - Developing User Interfaces for Microsoft Windows, E.N. McKay, Microsoft Press.
- Norman, 88 - "The design of everyday things", D. Norman, Basic Books. This book contains much information on how to design for anything to be used by humans--from physical objects to computer programs to conceptual tools.
- ISO 9241, 98 Ergonomic requirements for office work with VDTs- Dialogue principles, 1996.
- Ravden & Johnson, 89, Evaluation usability of human-computer interfaces: A practical method. - S.J. Ravden and G.I. Johnson Ellis Horwood Limited, New York, 1989
- Polson & Lewis, 90, Theory-based design for easily learned interfaces. Holcomb & Tharp Polson, P. G. and Lewis, C. H. Human-Computer Interaction, vol. 5, p191-220, 1990
- Holcomb & Tharp, 91. What users say about software usability. Holcomb, R. and Tharp Int. Journal of Human-Computer Interaction, 3(1):49--78

3.2 Survey of Design Heuristics and Guidelines for Usability from Literature

The information for this survey comes largely from the same sources that were considered in the previous section when surveying usability attributes.

The only new source considered is [Baecker, 95] which is a compilation of readings which have been fundamental in the field, along with extended introductions to each issue. In the chapter dedication to Design and Evaluation there is a survey of design principles which is used in this document in sections 3.2.6 to 3.2.8.

3.2.1 [Nielsen, 93]

Nielsen defines the following design heuristics

- Visibility of system status: The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.
- Match between system and the real world: The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.
- User control and freedom: Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.
- Consistency and standards: Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.
- Error prevention: Even better than good error messages is a careful design which prevents a problem from occurring in the first place.
- Recognition rather than recall: Make objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.
- Flexibility and efficiency of use: Accelerators -- unseen by the novice user -- may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.
- Aesthetic and minimalist design: Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.
- Help users recognise, diagnose, and recover from errors: Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.
- Help and documentation: Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

3.2.2 [Constantine, 99]

Constantine and Lockwood present in Appendix B their “Eleven Ways to Make Software More Usable: General Principles of Software Usability”. They separate five *rules* of usability from six more concrete *principles* of usability.

Five Rules of Usability:

1. Access Rule: The System should be usable, without help or instruction, by a user who has knowledge and experience in the application domain but no prior experience with the system.
2. Efficacy Rule: The system should not interfere with or impede efficient use by a skilled user who has substantial experience with the system.
3. Progression Rule: The system should facilitate continuous advancement in knowledge, skill, and facility and accommodate progressive change in usage as the user gains experience with the system.
4. Support Rule: The system should support the real work that users are trying to accomplish by making it easier, simpler, faster, or more fun or by making new things possible.

5. Context Rule: The system should be suited to the real conditions and actual environment of the operational context within which it will be deployed and used.

Six Principles of Usability:

1. Structure Principle: Organize the user interface purposefully, in meaningful and useful ways that put related things together and separate unrelated things based on clear, consistent models that are apparent and recognisable to users.
2. Simplicity Principle: Make simple, common tasks simple to do, communicating clearly and simply in the user's own language and providing good shortcuts that are meaningfully related to longer procedures.
3. Visibility Principle: Keep all needed tools and materials for a given task visible without distracting the user with extraneous or redundant information: What You See Is What You Need (WYSIWYN).
4. Feedback Principle: Through clear, concise, and unambiguous communication, keep the user informed of actions or interpretations, changes of state or condition, and errors or exceptions as these are relevant and of interest to the user in performing tasks.
5. Tolerance Principle: Be flexible and tolerant, reducing the cost of mistakes and misuse by allowing undoing and redoing while also preventing errors wherever possible by tolerating varied inputs and sequences and by interpreting all reasonable actions reasonably.
6. Reuse Principle: Reduce the need for users to rethink, remember, and rediscover by reusing internal and external components and behaviors, maintaining consistency with purpose rather than merely arbitrary consistency.

3.2.3 [Shneiderman, 98]

Shneiderman presents three main principles, one of which relates to eight rules for interface design, which are close to the notion of usability heuristics defined by other authors.

- Principle 1: Recognise diversity. Before beginning a design we must make the characterisation of the users and the situation as precise and complete as possible.
- Principle 2: User the Eight Golden Rules of Interface Design.
  - Strive for consistency. Consistent sequences of actions should be required in similar situations; identical terminology should be used in prompts, menus, and help screens; and consistent colour, layout, capitalisation, fonts and so should be employed throughout.
  - Enable frequent users to use shortcuts. Abbreviations, special keys, hidden commands and macro facilities.
  - Offer informative feedback. For every user action, there should be system feedback.
  - Design dialogs to yield closure. Sequences of actions should be organised into groups with a beginning, middle, and end. The aim is to provide the user with a sense of accomplishment.
  - Offer error prevention and simple error handling. As much as possible, design the system such that users cannot make a serious error. Erroneous actions should leave the system unchanged, or the system should give instructions about restoring the state.
  - Permit easy reversal of actions. The units of reversibility may be a single action, a data-entry mask, or a complete group of actions.
  - Support internal locus of control. Users should feel they are in control of the system.

- Reduce short-term memory load. Displays should be kept simple, multiple page displays be consolidated, window-motion frequency be reduced, and sufficient training time be allotted for codes, mnemonics, and sequence of actions. Where appropriate, online access to command-syntax forms, abbreviations, codes, and other information should be provided.

- Principle 3: Prevent Errors.
  - Correct matching pairs. This rule is directed to avoid syntactical errors when writing in a programming language or similar. It can be also applicable to word processors.
  - Complete sequences. Group different steps which are often used jointly. Example: The sequence of dialing up, setting communication parameters, logging on, and loading files. Another example: In a word processor, setting the alignment, the font, the font size and the letters in uppercase for section titles; it should not be necessary to set those characteristics every time a section title is entered.
  - Correct commands. Automatic command completion for command languages, and limiting the possible choices to permissible commands in a more developed interface.

Finally, Shneiderman offers some additional guidelines for data display and for data entry. These are not detailed in this document, as they refer to the main principles mentioned above, and their application to specific elements in the user interface.

3.2.4 [Hix, 93]

Hix and Hartson's guidelines on user interaction design are classified into twelve different areas. For some of them there are several related guidelines. They are presented according to the authors' distribution.

- User-Centred Design:
  - Practise user-centred design.
  - Know the user.
  - Involve the user via participatory design.
  - Prevent user errors.
  - Optimize user operations (accelerator keys, macros, abbreviations, etc.).
  - Keep the locus of control with the user.
  - Help the user get started with the system.
- System model:
  - Give the user a mental model of the system, based on user tasks. A system model sets the architectural framework for a system. It typically is device-, data-, and operation-oriented and represents flow of data and operations performed on those data. This maps into a conceptual model (the one the developer offers to the user). This, in turn, translates into a user's mental model, which is how a user perceives the system. Developing a good system model is very important during interaction design.
- Consistency and simplicity:
  - Be consistent.
  - Keep it simple.
- Human memory issues:

- Account for human memory limitations by giving the user frequent closure on tasks. The famous "seven plus or minus two chunks". Let the user recognize, rather than having to recall, whenever feasible.

- Cognitive issues:
  - Use cognitive directness. Cognitive directness involves minimizing mental transformations that a user must make. I.e., it is better to use "Command-c" for the "cut" command than a meaningless sequence such as "Esc-F7".
  - Draw on real-world analogies.
- Feedback
  - Use informative feedback. There are two types of feedback: Articulatory and semantic. Articulatory feedback tells users that their hands worked correctly (that the intended menu was the one actually selected), while semantic feedback tells them that their heads worked correctly (that the intended menu was the *right* menu to choose for the task at hand).
  - Give the user appropriate status indicators.
- System messages
  - Use user-centred wording in messages.
  - Use positive, nonthreatening wording in error messages.
  - Use specific, constructive terms in error messages.
  - Make the system take the blame for errors.
- Anthropomorphisation
  - Do not anthropomorphise. The point is that anthropomorphisation is very easy to do badly.
- Modality and reversible actions
  - Use modes cautiously.
  - Make user actions easily reversible.
- Getting the user's attention
  - Get the user's attention judiciously. Avoid excessive underlining, bold face, inverse video, blinking, audio, etc.
- Display issues:
  - Maintain display inertia. A good interaction design changes as little as possible from one screen to the next.
  - Organize the screen to manage complexity.
- Individual user differences
  - Accommodate individual user experiences and differences.
  - Accommodate user experience levels.

3.2.5 [Preece, 94]

Chapter 24 of this work addresses the issue of guidelines. It distinguishes high level principles from low level detailed rules. The former correspond to the concept being dealt with in this document.

This work presents and exemplifies four principles:

- Know the user population
- Reduce cognitive load
- Engineer for errors
- Maintain consistency and clarity

3.2.6 (Hansen, 1971) as cited in [Baecker, 95]

- Know the user
- Minimise memorisation
- Optimise operations
- Engineer for errors

3.2.7 (Rubinstein and Hersh, 1984) as cited in [Baecker, 95]

These authors present a list of 93 design principles, including

- Designers make myths; users make conceptual models
- Separate design from implementation
- Describe before you leap
- Develop an explicit use model
- Maintain a consistent myth
- Make states visible and visibly distinguished
- Minimise conceptual load
- Respect the rules of human conversation
- Interrupt with care
- Respond with an appropriate amount of information
- Do not misrepresent capabilities
- Do not violate the rules of human conversation
- Be consistent in the use of language
- Teach by example, not by formalism
- Avoid arbitrary syntax
- Use application terminology
- Use standard language
- Treat learning aids as part of the system
- Write the user's guide first
- Support clear conceptual models for documentation
- Examples, examples, examples
- Provide an easy way out

- No surprises
- Don't blame the user
- Make it easy to correct mistakes
- Articulate the evaluation goals
- Select subjects who are unfamiliar with the system
- Select subjects who are representative of the target population
- Use representative tasks
- Videotape real users

3.2.8 (Heckel, 1991) as cited in [Baecker, 95]

This book is structured around thirty design elements:

- Know your subject
- Know your audience
- Maintain the user's interest
- Communicate visually
- Leverage the user's knowledge
- Speak the user's language
- Communicate using metaphors
- Focus the user's attention
- Anticipate problems in the user's perception
- If you can't communicate, don't do it
- Reduce the user's defensiveness
- Give the user control
- Support the problem-solving process
- Avoid frustrating the user
- Help the user cope
- Respond to the user's actions.
- Don't let the user focus on mechanics
- Help the user to crystallise his thoughts
- Involve the user
- Communicate in specifics, not generalities
- Orient the user in the world
- Structure the user's interface
- Make your product reliable
- Serve both the novice and the experienced user.
- Develop and maintain user rapport

- Consider the first impression
- Build a model in the user's mind
- Make your design simple
- But not too simple.
- You need vision!!!

3.2.9 [Scapin, 97]

Scapin defines the following ‘ergonomic principles’

- Guidance
- Prompting
- Grouping and distinguishing items
- Grouping by location
- Grouping by format
- Immediate feedback
- Legibility
- Workload
- Brevity
- Conciseness
- Minimal actions
- Information density
- Explicit control
- Explicit user action
- User control
- Adaptability
- Flexibility
- User’s experience
- Error management
- Error protection
- Quality of error messages
- Error correction
- Consistency
- Significance of codes
- Compatibility

3.2.10 [Mckay, 99]

The principles give an indication of the kind of problems and questions (McKay 1999) users may have when interacting with a system.

- 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 doing correctly.

These principles assume that users always exhibit fully rational behavior. In practice, users make mistakes and do things they do not really wanted to do. Therefore, additional principles are:

- 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

3.2.11 [Norman, 88]

Norman defines the following user problem categories:

- Visibility
- Affordance
- Natural Mapping
- Constraints
- Conceptual Models
- Feedback
- Safety
- Flexibility

3.2.12 [ISO 9241-10, 96]

ISO 9241-10 provides the following dialogue principles.

- Suitability for the task
- Self-descriptiveness
- Controllability
- Conformity with user expectations
- Error tolerance

- Suitability for individualisation
- Suitability for learning

3.2.13 [Ravden & Jonson, 89]

Ravden & Johnson evaluate usability using the following criteria

- visual clarity
- consistency
- compatibility
- informative feedback
- explicitness
- appropriate functionality
- flexibility and control
- error prevention and correction
- user guidance and support

3.2.14 [Polson & Lewis, 90]

Polson and Lewis suggest the following principles for the design of intuitive systems, which improves usability to a certain extent.

- Make the repertoire of available actions salient.
- Use identity cues between actions and user goals as much as possible.
- Use identity cues between system responses and user goals as much as possible.
- Provide an obvious way to undo actions.
- Make available actions easy to discriminate.
- Offer few alternatives.
- Tolerate at most one hard-to-understand action in a repertoire.
- Require as few choices as possible.
-

3.2.15 [Holcomb & Tharp, 91]

Holcomb and Tharp present the following principles for design for successful guessing.

- Able to accomplish the task for which the software is intended
- Perform tasks reliably and without errors.
- Uniform command syntax
- Consistent key definition throughout
- Show similar information at the same place on each screen

- Learnable through natural, conceptual model
- Contains familiar terms and natural language
- Provide status information
- Don't require information entered once to be reentered
- Provide lists of choices and allow picking from the lists
- Provide default values for input fields
- Prompt before destructive operations
- Show icons and other visual indicators
- Immediate problem and error notification
- Messages that provide specific instructions for actions
- On-line help system available
- Informative, written documentation
- Ability to undo results of prior commands
- Ability tore-order or cancel tasks
- Allow access to operations from other applications/operating system from within the interface

3.3 Industry Vision of Usability

This section reflects the main items to be considered when developing usabile systems from a point of view of the industrial partners of the STATUS consortium, IHG and LogicDIS. The practical experience of such partners is presented in the sections below.

3.3.1 IHG Experience

The IHG has acquired knowledge of usability in several departments (graphics design, technology and consultancy) while working on, among other things, corporate web sites, B2B platforms, financial applications and e-learning applications.

3.3.1.1 Basic Usability Characteristics

Largely, industrial experience agrees with the descriptions of the usability attributes set out in the previous section, especially in the case of learnability, memorability, efficiency, satisfaction and error rate. However, there is another aspect of software that could be proposed as being part of usability. This aspect is predictability.

Predictability is related to learnability, but some subtle differences should be noted. Once a user has learned to use all of the functions of an application, they know what will happen as a result of each action. This is due to the learning experience. If an application is predictable, this means that the application will works as the user expects (they will be able to predict the result of an action) without having previously learned all of the application's funcitons and behaviours.

Example:

Purchasing a book: The buying process consists of several steps (finding the book, agreeing the price, entering customer data, entering delivery data, final agreement…). The user expects that at the end of the process they will be presented with a final confirmation screen with a summary of the transaction

before their credit card is charged. If the screen is not shown, the system would have, in some way, a lower degree of usability because it will not work as the user expected.

Another point of view is that an application must be predictable in the sense that all of the procedures work in consistent way. Therefore when the user uses a new feature, the results will be as they expected.

Example:

Imagine a complex application where the help information for each procedure is accessed and presented in a different way. This will be very confusing for the user. On the other hand, if the help is accessed from the same menu and icon and the help information is displayed using a particular look and feel, the user will understand it better even though the user has never used the procedure before.

3.3.1.2 Other Aspects

Apart from predictability, there are other aspects that industry would discuss if they are part of usability, although these aspects usually are not included in the typical decomposition of usability attributes.

Accessibility

This point is important when considering disabled users. An application must provide mechanisms to enable such users to use the system.

Another way of looking at accessibility is that using different methods or tools to access and interact with it might alter the general usability of an application.

Example:

If the application is a catalogue, the basic functionality is searching and looking at the items in the catalogue. The catalogue can be accessed using different tools, a web browser, a WAP phone, a particular application etc. The user will choose between the different tools depending on diverse factors (the hardware, the operating system, the bandwidth of the internet connection, …) but the functionality is the same for all tools.

Therefore, it could be said that the global usability of the system is better if there are several channels (tools) to access the functionality.

Target

The identification of the target group of users for an application can be used to improve the usability of that application.

Example:

In a commercial web site, if you identify that a group of users visit your site to compare your products with other vendors' products, it is important to provide a shortcut to the products in the homepage for better use of the site.

The target group identification can be linked with Personalisation topic. The interface can be adapted (personalized) to user profiles for better use.

Example:

In a complex financial graphic application, the application can ask for user confirmation to guide the beginner users and omit all confirmation questions for experienced users.

Information Screen Distribution

This is the way the information is distributed and displayed on the screen. This factor is very important in terms of usability. Additionally, the look and feel factors (font types and sizes, color schemes, …) are also essential.

Resistance to Change

This is the degree of resistance exhibited by users when faced with changes or evolution of functionality already adopted and learned.

Being a subjective sensation, this is difficult to measure, but can be translated into more concrete parameters such as cost of adoption, efficiency, etc.

3.3.2 LogicDIS Experience

In the following section we identify a set of usability attributes and design guidelines that are of high importance in terms of the overall software quality from the user's point of view. These attributes are strongly related to software architecture and most of them are difficult to improve or support if the software was designed without taking them into account.

The attributes are divided in five main categories: Consistency - Similarity, Flexibility, Simplicity, Errors and Completeness.

3.3.2.1 Consistency – Similarity

Uniformity

Similar tasks (from user's point of view) must demand (whenever possible) the same user interaction style.

Example: A user may expect that the registration task of two (even completely different) products in a company's site must demand from him the same interaction.

Undo

After the "undoing" of a task, the state of the system (from user's perspective) must be the same as it was before this task.

Example: A user has added some products to a shopping basket and the system marks those products to show him that they are already chosen, so as not to need to view the basket contents all the time. If later the user decides to remove all the items of a product from the shopping basket, he expects to see this product unmarked as if he had never selected it. But, when the user removes only some items of a product, he expects to continue seeing this product as marked.

Access to functionality from different entry points

Regardless of the entry point to a task, the entire set of necessary steps to complete it must be always available to the user (novice or expert).

Example: Usually, on a website, there are many ways (menus, icon-buttons, links etc) for a user to reach the ordering task. The expert users may choose a shorter route (icon-button) than the novices (menu). The system, based on these preferences, may decide to hide some steps (different to each case) in order to skip advanced details that may be confusing for a novice users, or to hide unnecessary details from expert users in order to increase their performance.
In any case the total set of the steps for ordering must be available to all users regardless of the entry point to the ordering task.

Consistency between versions - Backwards compatibility

If the interaction required to complete a task varies from version to version of a piece of software, often users have difficulties accomplishing tasks they already knew how to do with a previous version of the system.

Example: When a frequently used website, such as one for obtaining stock exchange information, changed to a newer version, many users who had used it in the past had difficulties adapting to this change as the required interaction style differed from the previous one.

3.3.2.2 Flexibility

Personalisation

How easily software can be configured to match the style and the preferences of the end user.

Example: Many web sites (e.g. “portals”) can support configuration of the contents that a user likes to see every time they log on to that web site, or they can arrange the website's look and feel according to their own needs and taste.

Example: Desktop applications may provide the option to customise toolbars or support shortcuts to frequently used tasks etc.

Memorability from the software's point of view - Adaptability

Although the term memorability is usually used to describe how easy it is for the user to use the software after a period of time, we believe that the software also should exhibit a certain degree of memorability which means that it should somehow remember a user’s personal preferences and style.

Example: The List of Recent tasks - processes (and recent documents mainly for desktop applications).

Addressability

This attribute is stronlgy related to the underlying software architecture and it has to do with how easy (and consistent) is to exploit the software's functionality from a wide range of "clients" i.e. from ta desktop PC, through the web, over WAP, through interactive TV, from a GPRS mobile, etc.

Example: Weather forecasts and stock exchange information are good examples of the kinds of information a user wants to access from different devices. E-commerce to mobile commerce is another issue that a system’s designers must take to account in order to maximise its usability.

Internationalisation - Multilingual

Often internationalisation, even for applications intended to domestic market, is a very useful capability and in some cases essential.

3.3.2.3 Simplicity

Self - explanatory / documented

If on screen labels are self-explanatory, users rarely need to search a user's guide in order to accomplish some tasks, so they help a user to feel the system is simple to use.

Less mandatory user interaction

The requirement for a large set of mandatory entries in order to accomplish a task often makes the user feel that the system is complicated. To prevent this, the designer must keep this set of mandatory interactions as small as possible

Example: Wizards with many steps must have a few mandatory first steps and then support something like a 'Finish' button to skip the other optional steps with some default values that can be changed later.

Example: Many web sites require, for one reason or another (marketing statistics etc), a large amount or information from users before letting them complete an action (download softweare etc). This

sometimes leads to the loss some visitors because they think that the specific system needs too much effort from the user for the services it provides.

Fast access to common tasks

The tasks that are most commonly performed using a system will vary from user to user. The system has to be designed with short paths to the most common tasks for the majority of users.

The system then has to adapt to each user's set of common tasks and give the ability of shortening the paths to those tasks separately.

Example: If a site can log the most popular tasks, then it may notice for example that the sports news pages are more popular than stock exchange information. After that, it can shorten the route to sports news by adding link on the first page, improving the performance for the users who in the past had to search for it.

3.3.2.4 Errors

Error Handling

When an error state error is encountered, one safe option from a user perspective is to put the system into an earlier safe state as if error not be happened.

This state must not be very far from the point where the error was encountered in order to ensure no loss of the user's work.

Example: When a user is trying to accomplish a task and in the seventh step an error occurs, it is very helpful (and logical) for him to return to the previous step (and not to the first) and to continue from there trying again to avoid the error.

Meaningful and accurate messages

Error messages must be meaningful to user (and not only to software engineers only), so that users realise what has happened.

Example: The message "Please select another user name. This name is already used by another user" obviously better than the message "Unique key [K_12] constraint violation".

There should be suggestions to the user in order to help them overcome a system error. A lack of suggestions may well drive the user to a dead end.

User Errors

Incorrect data entries

Incorrect data entries are the most frequent type of user error. Often users feel that they own the data they have inserted into a system, so the system should let the user modify almost any of the data have entered at a later date.

Incorrect use of a task (ambiguity)

This type of error is not so prolific as the previous one but may be more destructive. The system has to provide the user with mechanisms like 'undo' or 'cancel' in order to make the user feel safe when trying new functions.

Field vs. Form vs. Process Validation

Error messages must be presented to users as soon as an error is encountered in order to avoid loss of further input.

Example: When a user fills a registration or ordering form and the input in one field is invalid, the best thing the system can do is to inform the user as soon as possible, i.e. as soon as the field is completed. Some systems have form validation instead of field validation, so the user will be informed about the problem after the form submitted. Even worse, the user be informed after the end of the whole process.

3.3.2.5 Completeness

Incomplete functionality

Often the functionality that supports some tasks is not implemented completely in a certain version of a piece of software, or is developed with diminished functionality. This may cause users to misjudge the system as a whole.

Incomplete update

Some systems have a complete infrastructure and functionality based on this infrastructure, but little, or poorly updated content data.

Example: A portal may have a very strong infrastructure to support live updates of the contents of every theme it covers. Among other things it can provide information about the clubs in a town, but there are no address for some of them or few registered clubs.

3.4 Usability Properties considered in the STATUS Project

As can be note in previous sections between different classifications of usability heuristics, and design principles there is a lot of overlap, as well as there exists such overlap between the information gathered form literature and information provided by industrial partners. By step by step analysis and discussion the STATUS parnters led to the following list of usability properties; notice that some of the usability properties are divided in sub properties.

- Providing feedback
- Error management
    - Error prevention
    - Error correction
- Consistency
    - UI consistency
    - Functional consistency
- Guidance
- Minimize cognitive load
- Explicit user control
- Natural mapping
    - Predictability
    - Semiotic significance
    - Ease of navigation
- Accessibility

- Disabilities
- Multi-channel
- Internationalisation

Below we give a short definition for each usability property, to avoid confusion with other authors' definitions.

3.4.1 Providing feedback

The system should provide at every moment feedback to the user so he or she can know what is going on, that is, what the system is doing at every moment. Keep in mind that feedback and to provide feedback are different things.

3.4.2 Error management

The system should provide a way to manage errors. This can be done in two ways:

- By preventing errors to happen users make no or less mistakes.
- By providing an error correcting mechanism we can correct errors made by users.

3.4.3 Consistency

Users should not have to wonder whether different words, situations, or actions mean the same thing. It is regarded as an essential design principle that we should use consistency within applications. Consistency makes learning easier because we have to learn things only once, because next time the same thing is faced in another application, it is familiar. Visual consistency increases perceived stability which increases user confidence in different new environments. Consistency might be provided in different ways:

- User interface elements should be consistent as well in aspect and structure.
- Functional consistency, that is, the way of performing different tasks across the system should be consistent, but also with other similar systems, and even between different kinds of applications in the same system.

3.4.4 Guidance

In order to help the user understand and use the system, we should provide informative, easy to use and relevant guidance and support as well in the application as in the user manual.

3.4.5 Minimize cognitive load

Systems should minimize the cognitive load e.g. Humans have cognitive limitations, systems should have this limitations in mind.

3.4.6 Explicit user control

It is a design principle that we should support "direct manipulation"; e.g. the user should get the impression that he is "in control" of the application. Interaction is more rewarding if the users feel that they directly influence the objects instead of just giving the system instructions to act.

3.4.7 Natural mapping

The system should provide a clear relationship between what the user wants to do and the mechanism for doing it. This property can be structured as follows:

- Predictability: The system should be predictable; e.g. to the user the behavior of the system should be predictable
- Semiotic significance: Systems should be semiotic significant; Semiotics, or semiology, is the study of signs, symbols, and signification. It is the study of how meaning is created, not what it is.
- Ease of navigation: Systems should be easy to navigate.

3.4.8 Accessibility

Systems should be accessible in everyway that is required. Such property might be descomposed as follows:

- Disabilities: Systems should provide support for people that are disabled (blind/deaf/short sighted);
- Multi-channel: The system should be able to support access via various media; Multi channeling (accessing) in this way is a very broad concept varying from being able to browse a website via a phone or being able to auditory browse a website (support for audio output).
- Internationalisation: Systems should provide support for internationalisation, because users are more familiar with their own language, currency etc.

4. USABILITY PATTERNS

One of the products of the research on this project into the relationship between software architecture and usability is the concept of a usability pattern. We have chosen to use the term "usability pattern" to refer to a technique or mechanism that can be applied to the design of the architecture of a software system in order to address a need identified by a usability property at the requirements stage (or an iteration thereof). Other authors have previously coined the term usability pattern, and there is some overlap with the way that we use it here, but there are also some differences. Most significantly we are only considering patterns which can be applied during the design of a system's architecture, rather than later during the detailed design stage. There is not a one to one mapping between usability patterns and the usability properties that they affect. A pattern may be related to any number of properties, and each property may be improved (or impaired) by a number of different patterns. The choice of which pattern to apply may be made on the basis of cost and the trade off between different usability properties, or between usability and other quality attributes such as security or performance.

Some previous work has been done in the area of usability patterns, by Tidwell [Tidwell, 99] and van Welie [van Welie, 2000]. This work has been informed by their work, but takes a different standpoint, concentrating on the architectural effect that patterns may have on a system. Bass and John [Bass et al, 2001] give examples of architectural patterns that may aid usability, and their work has also been useful for this initial study.

A usability pattern is not the same as a design pattern (it is not a design pattern that, if employed, will affect usability) as the pattern does not specify implementation details in terms of classes and objects, it is a level abstracted from that. It may be possible to use a number of different methods to implement the solution presented in each usability pattern. In each case, it is only presented that the decision to use a certain pattern will have some affect on the usability of the system, and will also influence the software architecture.

One thing that usability patterns share with design patterns is that their goal is to capture design experience in a form that can be effectively reused by software designers in order to improve the usability of their software, without having to address each problem from scratch. The aim is to take what was previously very much the "art" of designing usable software and turn it into a repeatable engineering process.

Another aspect shared with design patterns is that a usability pattern should not be the solution to a specific problem, but should be able to be applied in order to solve a number of different problems in a number of different systems, in accordance with the principle of software reuse.

As well as these patterns, which can be applied to the architecture of a piece of software, it has been noted that there are some techniques that can be applied to the way that the development team designs and builds the software that may lead to improvements in usability for the end user. For instance, the use of an application framework as a baseline on top of which to construct applications may be of benefit, promoting consistency in the appearance and behaviour of components across a number of applications. For instance, using the Microsoft Foundation Classes when building a Windows application will provide "common" Windows functionality that will be familiar to users who have previously used other applications build on top of this library. This is not a pattern that can be applied to the architecture in the same way as those discussed in the next pages, but it is nonetheless something which will be considered during the further study of the relationship between software architecture and usability during the remaining parts of this project, as we are also interested in the process by which software is developed.

The following sections give a list of examples of different usability patterns. For each pattern, the following are given:

- **Name** – indicative of its purpose
- **Description** – a brief description of the mechanism or technique
- **Relationship with Software Architecture** – outlines the architectural impact of the pattern.
- **Relationship with Usability Properties** – identifies the usability properties the pattern relates to.
- **Example** – an example of the use of the pattern in a system that is in use.

This list is not intended to be exhaustive, and it is envisaged that future work on this project will lead to the expansion and reworking of the set of patterns presented here, including work to fill out the description of each pattern to include more of the sections which traditionally make up a pattern description, for instance what the pros and cons of using each pattern may be.

4.1 Progress Indication

Description

A progress indicator is a mechanism that can be used to indicate how much of the current task has been completed and how long it will take to finish. This allows the user to see that the computer is doing some work, and has not just “hung”.

Relationship with Software Architecture

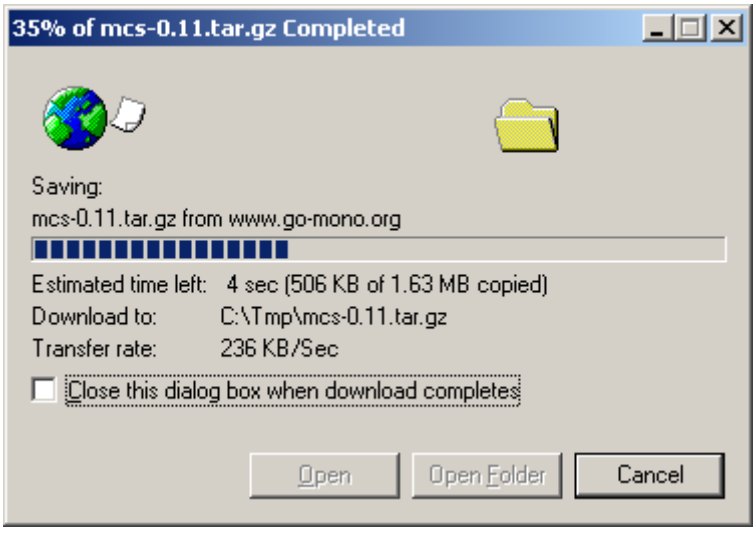
It must be possible for the process that generates the progress indicators to run concurrently with the process whose progress is being displayed. The system must contain a component that enables it to create a model of the tasks that it is carrying out so that it can judge progress and the amount of work that remains to be done.

Relationship with Usability Properties

Giving an indication of progress provides feedback to the user about what the system is currently doing.

Example

Progress bars are commonly displayed by web browsers during the download of a file.



Downloading a file with Internet Explorer

4.2 Alerts

Description

An alert is a message from the system to the user that a change of state has occurred that the user ought to know about.

Relationship with Software Architecture

To support the provision of alerts to the user, there needs to be component that monitors the behaviour of the system and sends messages to an output device.

Relationship with Usability Properties

Alerts help to keep the user informed about the state of the system.

Example

If a new email arrives, the user may be alerted by means of an aural or visual cue.

If a user makes a request to a webserver that is currently off line, they will be presented with a popup window telling them that the server is not responding.

4.3 Status Indication

Description

The user should be provided with information pertaining to the current state of the system.

Relationship with Software Architecture

To support the provision of status information to the user, there needs to be component that monitors the behaviour of the system and sends a message to an output device.

Relationship with Usability Properties

Giving an indication of the system's status provides feedback to the user about what the system is currently doing, and what will result from any action they carry out.

Example

The status bar at the bottom of the screen in Microsoft Word shows the current page number, the position of the cursor on the screen in rows and columns, whether certain modes, such as overwrite, are currently active, and the current language.

4.4 History Logging

Description

Recording a log of the actions that the user (and possibly the system) takes allows the user (or the system) to look back over what was done previously.

Relationship with Software Architecture

In order to implement this, a repository must be provided where information about actions can be stored. Consideration should be given to how long the data is required for. Actions must be able to be represented in a suitable way for recording in the log.

Relationship with Usability Properties

Providing a log helps the user to see what went wrong if an error occurs and may help them to correct that error. Being able to refer to actions that were carried out previously may help with "recognition rather than recall".

Example

Web browsers create a history file detailing all the websites that the user has visited. Databases typically write a log of the transactions which are completed.

4.5 Undo

Description

The ability to undo an action and return to the previous state.

Relationship with Software Architecture

In order to implement undo, a component must be present that can record the sequence of actions carried out by the user and the system, and also sufficient detail about the state of the system between each action in order that the previous state can be recovered.

Relationship with Usability Properties

Providing the ability to undo an action helps the user to correct errors if they make a mistake. It helps the user to feel that they are in control of the interaction.

Example

Microsoft Word provides the ability to undo and redo (repeatedly) almost all actions that the user can carry out while working on a document.

4.6 Form or Field Validation

Description

If a user is entering multiple items of data on one screen, it is possible to check that each field contains valid data either all at once when the "submit" or "ok" button is pressed (form validation), or individually each time a data item is entered (field validation). With form validation it may be the case that one invalid entry leads to the whole form having to be filled in again.

Relationship with Software Architecture

In a web situation, form versus field validation often equates to doing the validation check on the server or on the client respectively.

Relationship with Usability Properties

This pattern relates to a provision for the management of errors.

Example

There techniques are often employed in forms on websites where the user has to enter a number of different data items, for example when registering for a new service, or buying something.

4.7 Preview

Description

A user may wish to see what the results of an action (possibly a resource consuming one) will be before executing the command.

Relationship with Software Architecture

The architecture must provide the ability to synthesise the results of an action (e.g printing) in the software before actually performing the action.

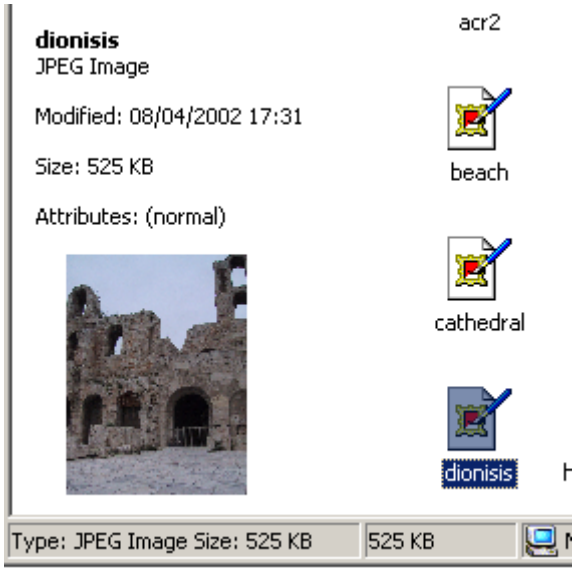
Relationship with Usability Properties

Providing the ability to preview the results of an action helps to prevent against error.

Example

When opening photos from a digital camera named 1.jpg, 2.jpg etc it is useful to see a thumbnail of the image before loading it, as it is then much easier to make sure that it is the right picture.

When printing a document, it is useful to be able to preview the layout before commiting it to paper.



Previewing photos in Windows file explorer.

4.8 Model/View/Controller Separation

Description

Separating the model of the data from the way that it is displayed facilitates creating different views for different uses and users. Separating the controller component allows different types of interface or device to be used for controlling the interaction between the humand and the system.

Relationship with Software Architecture

The architecture must be constructed in such a way that components that hold the model of the data currently being processed are separated from components that are responsible for representing this data to the user and those that handle input events. The model component needs to notify the view component when the model is updated, so that the display can be redrawn.

Relationship with Usability Properties

Separating the model of the data from the view aids consistency across multiple views when these are employed.

Separating out the controller allows different types of input device to be used by different people, which may be useful for disabled users.

Example

Microsoft Word has a number of possible views that the user can select (normal view, outline view, print layout view…) and switch between at will, which all represent the same underlying data.

4.9 Emulation

Description

A system can be made to emulate the appearance and/or behaviour of a different system.

Relationship with Software Architecture

Command interfaces and views need to be replaceable and interchangable, or there needs to be provision for a translation from one command language and view to another in order to enable emulation.

Relationship with Usability Properties

Emulation can provide consistency in terms of interface and behaviour between different pieces of software.

Example

Microsoft Word 97 can be made to emulate WordPerfect, so that it is easier to use for users who are used to that system.

4.10 Workflow Model

Description

Modelliing workflow enables different users to be provided with only the tools or actions that they need in order to perform their specific task on a piece of data before passing it on the the next person in the workflow chain, who will perform a different task.

Relationship with Software Architecture

A component or set of connectors that model the workflow is required, describing where the data flows. A model of each user in the system is also required, so that the actions that they need to perform on the data can be provided for them (see 4.11 user profile).

Relationship with Usability Properties

Targetting the user interface specifically to each user, dependent on the task that they need to perform in the workflow minimises the user's cognitive load.

Example

LogicDIS have developed software which models workflow.

4.11 User Profile

Description

The software system biulds and records a profile of each user, so that specific attributes of the system (concerning the layout of the user interface, the amount of data or options to show etc) can be set and reset each time that a different user comes to use the system. Different users may have different roles, and require different things from the software.

Relationship with Software Architecture

A repository for user data needs to be provided. This data maybe added to or altered either by the user setting a preference, or by the system.

Relationship with Usability Properties

Providing the facility to model different users allows a user to express preferences.

Example

Many websites recognise different types of users (e.g. customers or adminstrators) and present different functionality depending on who is using the site.

Amazon.com builds detailed profiles of each of its customers in order that it can recommend products that it thinks the user might be interested in on the front page of the site.

A personalised page from Amazon.co.uk

4.12 User Modes

Description

A system can provide different modes corresponding to different feature sets required by different types of users, or by the same user when performing different tasks. There may be simple or advanced modes.

Relationship with Software Architecture

Depending on the mode, the same set of controls may be mapped to different actions, via different sets of connectors, or more or less user interface components may be displayed.

Relationship with Usability Properties

Having different modes allows personalistation of the software to the current user's needs.

Example

Winzip allows the user to switch between "wizard" and "classic" modes, where the wizard mode gives more guidance, but the classic mode lets the expert user work more efficiently.

Many websites have different modes for different people, normal users or administrators.

4.13 Shortcuts

Description

A shortcut allows an experienced user to activate a feature which may be hidden “under the surface” of the interface with one quick gesture.

Relationship with Software Architecture

To allow shortcuts, several different user interface gestures need to be able to be mapped on to the same action underneath.

Relationship with Usability Properties

The provision of shortcuts allow the system to match the user’s level of expertise. An experienced user will use the shortcut where a novice will navigate a longer path through the user interface, perhaps being subject to a greater degree of guidance.

Example

Almost all Windows applications provide keyboard shortcuts for commonly accessed items from menus.

Websites may provide “deep links” to pages many clicks away, on the front page if (expecially combined with a user profile) they expect the user to want to jump to that page based on previous experience.

4.14 Context Sensitive Help

Description

Context sensitive help monitors what the user is currently doing, and makes available documentation that is relevant to the completion of that task.

Relationship with Software Architecture

There needs to be provision in the architecture for a component that tracks what the user is doing at any time and targets a relevant portion of the available help.

Relationship with Usability Properties

The provision of contect sensitive help can give the user guidance.

Example

Microsoft Word includes context sensitive help. Depending on what feature the user is currently using (entering text, manipulating an image, selecting a font style) the Office Assistant will offer different pieces of advice (although some users feel that it is too forceful in its advice).

Depending upon what the mouse cursor is currently pointing to, Word will pop up a small description or explanation of that feature.

4.15 Wizard

Description

The wizard pattern presents the user with a structured sequence of steps for carrying out an operation and guides them through one by one. The task as a whole is separated into a series of more manageable subtasks. At any time the user can go back and change earlier steps in the process.

Relationship with Software Architecture

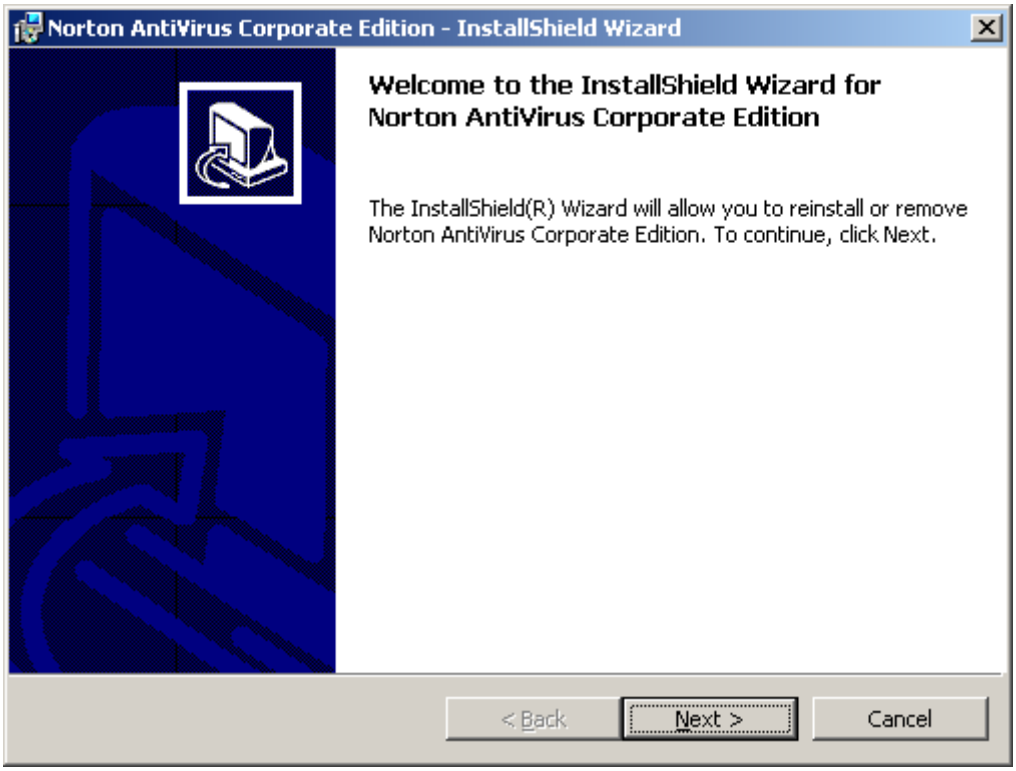
There needs to be provision in the architecture for a wizard component, which can be connected to other relevant components, the one triggering the operation and the one receiving the data gathered by the wizard.

Relationship with Usability Properties

The wizard helps with guidance, showing the user what each consecutive step in the process is.

Example

The install wizard used by most Windows programs guides the user through choosing various options for installation



4.16 Selection Indication

Description

The user and the system will often have one object or set of objects that have a special status. They are the objects that will be acted upon the next time that a command is issued. These objects should be indicated using some sort of highlighting.

Relationship with Software Architecture

There needs to be a component in the architecture that is responsible for monitoring which object or objects is currently selected. The component responsible for the view can then get this information and draw a suitable indicator in the display.

Relationship with Usability Properties

Allowing the user to change the selection at will enhances their feeling of control. Showing the currently selected object using some form of highlighting provides information to the user about the current state of the system.

Example

In a graphics package, for instance Corel Draw, the currently selected objects are highlighted by displaying their bounding boxes.

4.17 Cancel

Description

A user should be allowed to cancel a command that has been issued if they realise that they have done the wrong thing, to prevent reaching an error state. This is different to being able to undo an action after it has finished in order to return to the previous state.

Relationship with Software Architecture

There needs to be provision in the architecture for the component monitoring the user input to run independently from and concurrently with the components that carry out the processing of actions. The action processing components need to be able to be interupted.

Relationship with Usability Properties

Being able to cancel commands helps with error management, as if the user realises that they have done the wrong thing then they can interrupt and cancel an action before the error state is reached. It also gives the user the feeling that they are in control of the interaction.

Example

In most web browsers, if the user types a URL incorrectly, and the web browser spends a long time searching for a page that does not in fact exist, the user can cancel the action by pressing the “stop” button before the browser presents them with a “404” page, or a dialog saying that they server could not be found.

4.18 Multi-Tasking

Description

Multi-tasking describes the situation where the system (and the user) can manage several tasks at the same time, allowing switching between the different tasks as is most useful to completing work most efficiently and effectively.

Relationship with Software Architecture

A system should be designed so that it can be used along side any other system without interference. It may also be useful for the system to be able to manage more than one set of data at once, for example a word processor that can hold multiple documents open simultaneously). All of these things have architectural considerations.

Relationship with Usability Properties

Providing a multi-tasking environment aids the user's feeling of control over the system as at any point they can switch to the task that is of most interest to them.

Example

Windows is a multitasking environment which empowers the user to run a web browser showing a useful reference website while writing a document in a word processor, and then switch to check an email when one arrives.

Mircosoft Excel allows multiple spreadsheets to be opened at the same time without replicating the controls.

4.19 Macros

Description

It is often the case that the same sequence of commands or actions needs to be applied to a number of different objects. Providing the user with the ability to create macros or composite commands will help such a task to be completed more quickly and accurately as the command sequence can be triggered by one higher level action. Macros may be scripted using a form of programming language or they may be recorded.

Relationship with Software Architecture

Provision needs to be made in the architecture for commands to be grouped into composites, or for it to be possible to record and play back sequences of commands in some way. There needs to be an appropriate representation of commands, and a repostory for storing the macros.

Relationship with Usability Properties

Providing the ability to group a set of commands into one higher level command reduces the user's cognitive load, as they do not need to remember how to execute the individual steps of the process once they have created a macro, they just need to remember how to trigger the macro.

Example

All of Microsoft's office applications provide the ability to record macros, or to create them using the Visual Basic for Applications language.

Emacs allows the user to execute strings of commands which can be assigned to special key combinations.

4.20 Actions for Multiple Objects

Description

It is often the case that the same action needs to be applied to a number of different objects. Providing the user with the ability to group the objects and apply one action to them all "in parallel" will help such a task to be completed more quickly and accurately. If each object has to be treated individually, errors are more likely to be made.

Relationship with Software Architecture

Provision needs to be made in the architecture for objects to be grouped into composites, or for it to be possible to iterate over a set of objects performing the same action for each.

Relationship with Usability Properties

Providing the ability to perform the same action on a number of objects at once reduces the time that it will take the user to complete a task, as the system should be much faster at repeating actions than the human. The number of clicks (or equivalent actions) that the user has to make to complete the task is reduced.

Example

In a vector based graphics package such as Corel Draw, it is possible to select multiple objects and to perform the same action (change colour etc) on all of them at the same time.

5. SUMMARY & CONCLUSION

The following table summarises the different usability attributes, properties and patterns that have been considered in the previous sections of this document, and the relationships between them. In some circumstances it was felt that there was probably a link between, for example, one usability property and all of the usability attributes. To avoid the table becoming too cluttered, and the risk of possibly producing a fully connected graph, only the links thought to be strongest are indicated in the table. The relationships depicted in the table are an initial step in that they indicate potential relationships. Further work is required to substantiate these relationships and to provide models and assessment procedures for the precise way that the relationships operate, but even at this early stage it can be seen that all of the usabillity attributes are in fact linked to software architecture.

In conclusion, this working package has made a critical and important first step in providing a framework for systematically investigating the relationship between usability and software architecture. Patterns capture architectural concerns and usability properties provide a way forward in relating usability attributes to design decisions at the architectural level.

Usability Attributes	Usability Properties	Usability Patterns
Satisfaction	Provide feedback	Progress indication
		Alerts
Learnability		Status indication
	Error management	History logging
Efficiency	- error correction	Undo
	- error prevention	Form or Field validation
Reliability		Model/View/Controller separation
	Consistency	Emulation
	- user interface	
	- functional	
	- evolutionary	Workflow model
		Actions for multiple objects
Reliability	Minimising cognitive load	Macros
	Adaptability	User profile
Satisfaction	- matching user expertise	User modes
	- matching user preferences	Shortcuts
Learnability	Guidance	Context sensitive help
		Wizard
Efficiency	Explicit user control	Selection indication
	Natural mapping	Cancel
	Accessibility	Multi-tasking

		Model/View/Controller separation

6. REFERENCES

[Bass, 97] Bass, Clements, and Kazman. *Software Architecture in Practice*, Addison-Wesley 1997:

[Bass et al, 2001] L. Bass, B. John, J. Kates. *Achieving Usability Through Software Architecture*. CMU/SEI Technical report 2001

[Baecker, 95] R.M. Baecker, J. Grudin, W.A.S. Buxton, S. Greenberg. *Readings in Human-Computer Interaction: Toward the Year 2000*. Morgan Kaufmann, 1995.

[Bevan, 01] N. Bevan. "International standards for HCI and usability". *International Journal of Human-Computer Studies*, Vol. 55, No. 4, Oct 2001, pp. 533-552 . 2001.

[Bevan, 95] N. Bevan. "Measuring Usability as Quality of Use". Proc. of the 6th International Conference on Human-Computer Interaction. July, 1995.

[Boehm, 78] B. Boehm, J.R. Brown, H. Kaspar, M. Lipow, G.J. Macleod, M.J. Merritt. *Characteristics of Software Quality*. North Holland, 1978.

[Constantine, 99] L. L. Constantine, L. A. D. Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*. Addison-Wesley, New York, NY, 1999.

[Gould, 88] J. D. Gould. "How to Design Usable Systems" in *Handbook of Human-Computer Interaction*, edited by M. Helander. Elsevier, 1988.

[Hix, 93] D. Hix, H.R. Hartson. *Developing User Interfaces: Ensuring Usability Through Product and Process*. John Wiley and Sons, 1993.

[IEEE1061, 98] IEEE. *IEEE Std 1061: Standard for a Software Quality Metrics Methodology*. IEEE, 1998.

[ISO, 96] ISO 9241-10 *Ergonomic requirements for office work with VDTs*-Dialogue principles, 1996

[ISO9126, 91] ISO. *ISO 9126 Information Technology – Software quality characteristics and metrics.* ISO, 1991

[ISO9126, 00] ISO. *ISO 9126-1 Software Engineering – product quality – part 1: Quality Model*, 2000

[ISO9241, 98] ISO. *ISO 9241-11. Ergonomic Requirements for Office work with Visual Display Terminals. Part 11: Guidance on Usability*. ISO, 1998.

[ISO14598, 99] ISO/IEC. *ISO/IEC 14598-1, Software Product Evaluation: General Overview*. ISO/IEC , 1999.

[McKay, 99] E.N. McKay, *Developing User Itnerfaces for Microsoft Windows,* Microsoft Press

[Mayhew, 99] D. J. Mayhew. *The Usability Engineering Lifecycle*. Morgan Kaufmann, 1999.

[Nielsen, 93] J. Nielsen. *Usability Engineering*. AP Professional, 1993.

[Norman, 88] D. Norman, *The design of everyday things,* Basic Books

[Polson & Lewis, 90] Polson, P. G. and Lewis, C. H. *Theory-based design for easily learned interfaces.* Human-Computer Interaction, vol. 5, p191-220, 1990.

[Preece, 94] J. Preece, Y. Rogers, H. Sharp, D. Benyon, S. Holland, T. Carey. *Human-Computer Interaction*. Addison Wesley, 1994.

[Ravden, Johnson, 89] S.J. Ravden and G.I. Johnson*, Evaluation usability of human-computer interfaces: A practical method*. Ellis Horwood Limited, New York, 1989

[Rohlfs, 98] S. Rohlfs. "Transforming User-Centered Analysis into User Inteface: The Redesign of Complex Legacy Systems" in *User Interface Design*. ed. by L. E. Wood. pp. 185-214. CRC Press, 1998.

[Scapin, 1997] D.L.Scapin, J.M.C. Bastien, *Ergonomic criteria for evaluationg the ergonomic quality of interactive systems,* Behaviour & Inromation Technology, vol 16, no 4/5, pp.220-231

[Shackel, 91] B. Shackel. "Usability – context, framework, design and evaluation". In *Human Factors for Informatics Usability*. pp 21-38. Ed. by B. Shackel and S. Richardson. Cambridge University Press, 1991.

[Shneiderman, 98] B. Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley, 1998.

[Tidwell, 99] J. Tidwell. *Common Ground: A Pattern Language for Human-Computer Interface Design* http://www.mit.edu/~jtidwell/common_ground.html

[van Welie, 2000] M. van Welie, H. Troetteberg. *"Interaction Patterns in User Interfaces"* PLoP 2000

[Wixon, 97] D. Wixon, C. Wilson. "The usability Engineering Framework for Product Design and Evaluation". In *Handbook of Human-Computer Interaction*. pp. 653-688. Ed. by M. G. Helander et al. Elsevier North-Holland, 1997.