To appear in the Proceedings of the IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction, Ellivuori, Finland, August, 1992, Elsevier Publ.

Structuring the Space of Interactive System Properties

Gregory D. Abowd[a], Joëlle Coutaz[b] and Laurence Nigay[b]

[a] Dependable Computing Systems Centre and Human-Computer Interaction Group, Department of Computer Science, University of York, Heslington, York Y01 5DD, United Kingdom

[b] Laboratoire de Génie Informatique, BP 53 X, 38041 Grenoble Cedex, France

Abstract

We provide a structured classification of properties to guide the principled design of interactive systems. This classification is motivated by an existing software quality framework, which we extend with respect to the usability of the software product. We distinguish between high-level categories of product usability and criteria within those categories which can be represented and ultimately measured in terms of the software product itself. In this paper, we highlight three usability categories, learnability, interaction flexibility and interaction robustness, and define criteria which contribute to them.

Keyword Codes: H.1.2, H.5.2, D.2.10
Keywords: User/Machine Systems, User Interfaces, Software engineering Design

1. INTRODUCTION

A real challenge in the development of interactive software is the establishment of principled practices to ensure the quality of the products produced. Software quality assurance is an active area of research in software engineering. It is believed that the 1990s will be the quality era, in which software quality is quantified and brought to the center of the development process [1]. In this paper, we focus attention on software quality as it is affected by the interactive features of the software being developed. We can further divide this main objective of principled interactive software development into four more specific objectives:

- define general properties for interactive software;
- provide a method for organizing and assessing the interactive properties;
- provide formalisms in which to express more clearly the properties; and
- link these properties to software engineering practice and measurement.

We directly address the first two of these objectives, definition and organization, in this paper. In so doing, we provide the foundation for achieving the final two objectives in our further research.

There have been many attempts in the past to define the desirable properties for interactive software. This body of literature ranges from very specific interface guidelines for particular platforms, such as the Apple human interface guidelines [2], to very general principles of interactive behaviour, such as those given by Thimbleby [3, 4]. We desire a list of general properties that are not tied to any existing technology; indeed, we would hope that our list of properties might inspire new interaction mechanisms that satisfy a set of interactive properties not currently satisfiable with today's technology. Generality, however, is not enough, as we

also want to provide an organizational structure for interactive properties which can be readily extended. This paper is a preliminary attempt to provide such a structured catalogue.

An interactive property is a feature of an interactive system which is the subject of analysis and evaluation. Properties are neither necessarily good nor bad features; they simply serve to define an absolute design space which exhausts the possibilities for a design. The evaluation of a given design can occur at various stages in the design process. Currently, the majority of evaluation for usability of a system occurs in the form of user testing which takes place after implementation. User testing is essential, and development environments which allow for rapid prototyping and iterative design can maximize the effect of user testing by introducing them at earlier stages in the life cycle. Current research in HCI, however, is now making it plausible to provide an alternative to explicit user testing. Specifically, theoretical tools such as Barnard's Interactive Cognitive Subsystems (ICS) [5] and Young's Programmable User Model (PUM) [6] based on the SOAR problem solving architecture [7], make it possible to perform usability testing at earlier stages in the design process without the need for direct user testing. From our perspective as software engineers, the only way that this type of theoretical research will impact software evaluation is if the results of the HCI theory are expressed in software engineering terms. As we shall see in Section 2, this means that the factors affecting product usability must be characterized in terms of criteria for the software specification that can be measured by the software engineer.

Though we do not directly address the use of formalisms to represent the general interactive properties in this paper, there are two reasons why we promote their use. Firstly, a mathematical language provides the expressive precision necessary for intellectual debate. Secondly, since a significant portion of the software engineering community has recognized the need for formalism in design, the general purpose mathematical notations of software engineering, such as Z, VDM, CSP, CCS, Larch, RAISE, LOTOS and others, can provide the language needed to translate cognitive phenomena into their design equivalents. Our belief is that most, if not all, of the general interactive properties presented in this paper can be expressed as precise properties of the specification of an interactive system that can thus be evaluated before implementation. Indeed, in compiling the list of properties, we tended only to consider those which we felt were realizable within some formalism. More work must be done to substantiate this claim.

Methodologies exist for quality assessment in software engineering. We propose in the remainder of this paper to relate the organization of general interactive properties to one such quality assurance method, based on work by McCall and others [8, 9]. The McCall framework has directly inspired standards work done more recently by the IEEE and the AFCIQ (part of the French association for standards) [10, 11]. Though each of these earlier standards have considered the end user of the final product, those concerns have been underrepresented in the published standards. The McCall framework provides the structure for organizing general interactive properties in a way that is currently lacking.

Overview of paper

In Section 2, we briefly outline the McCall framework for the measurement of software quality, emphasizing how we intend to augment this framework by expanding the definition of the usability factor into three separate categories of learnability, interaction flexibility and interaction robustness. The following three sections provide a detailed discussion of software criteria which support each of those categories in turn, with the results of each section summarized in tabular form. We conclude in Section 6 with a summary of the contributions of this work and some suggestions for further research on the topic of quality measurement in interactive software design.

2. THE MCCALL FRAMEWORK

In this section, we will give a brief definition of the McCall framework for software quality measurement and explain how we can augment the framework by further discussion of the usability factor.

2.1. Definition

The process of quality assessment involves a contractual arrangement between the supplier (or vendor) and consumer (or customer) of a software product. The customer places requirements on the product they wish to use and it is up to the vendor to demonstrate that they have met those requirements. In order for the contract to make sense, therefore, the customer must be able to express requirements in a way that is suitable for both themselves and the vendor. To accommodate this, we distinguish between quality *factors* and quality *criteria* in order to represent the differing viewpoints of the customer and the vendor.

A quality factor is a software quality goal from the customer's perspective. They arise from a "management-oriented view of product quality." [9] These factors are separated into three categories that relate to a software product:

- operational characteristics (product operations);
- ability to support changes (product revision); and
- capacity to adapt to new environments (product transition).

There are 11 quality factors within these three categories, and these are defined more formally within the framework (see Table 1). Quality factors are not directly measurable. The idea in the McCall framework is that the customer is familiar with the meaning of the quality factors and, therefore, makes demands on the delivered product in terms of those factors.

A quality criterion is a property of the software product which a software engineer can directly measure at some stage in the development of the product. Each quality factor is related to some set of quality criteria which are believed to affect that factor. Precisely how the relationship between factors and criteria is generated empirically within the McCall framework is not our concern here. Rather, we are only interested in the distinction between high-level factors and measurable criteria. Within the scope of this paper, measurability of a criterion means the ability to represent its meaning within some formalism.

2.2. Improvements for interactive systems

Within the McCall framework, there has not been enough attention given to the identification of factors and criteria that affect the interaction between a human user and the computer system. The existing framework outlines 11 quality factors, only one of which, usability, directly concerns this interaction. The criteria related to the usability factor are:

- operability, i.e., ease of use;
- training, i.e., ease of learning; and
- communicativeness , i.e., ease of understanding.

In addition to ease of use, ease of learning, and communicativeness mentioned in the McCall model, the 1989 AFCIQ model expresses usability with self-descriptiveness (i.e., readability and the appropriateness of the documentation), consistency, completeness, and fault tolerance criteria. We argue that not only are the above descriptions of usability far too coarse to lead to a sound evaluation but also that usability in general is not given a high enough profile in the existing frameworks. It is understandable that these earlier frameworks do not give adequate attention to the usability factor. Research in HCI has only begun to mature over the past decade and very little of this research has trickled down into software engineering practice from where the original factors and criteria were gleaned. The definitions provided by McCall and others for software quality factors and criteria can be improved by input from the HCI research community.

Table 1
McCalls software quality factors

Category	Factor	Definition
Product operations	Correctness	Extent to which a program satisfies its specifications and fulfils the users mission objectives.
	Reliability	Extent to which a program can be expected to perform its intended function with required precision.
	Efficiency	The amount of computing resources and code required by a program to perform a function.
	Integrity	Extent to which access to software or data by unauthorized persons can be controlled.
	Usability	Effort required to learn, operate, prepare input, and interpret output of a program.
Product revision	Maintainability	Effort required to locate and fix an error in an operational program.
	Flexibility	Effort required to modify an operational program.
	Testability	Effort required to test a program to insure it performs its intended function.
Product transition	Portability	Effort required to transfer a program from one hardware configuration and/or software system environment to another.
	Reusability	Extent to which a program can be used in other applications-related to packaging and scope of the functions that programs perform.
	Interoperability	Effort required to couple one system with another.

In Table 2, we suggest an extension to the original McCall quality factors. We remove usability as a quality factor under the category of product operations and promote it to become a category by itself and add a further three quality factors which are directly linked to product usability: learnability; interaction flexibility; and interaction robustness. In the next three sections we expound on the definition of these usability factors and define related criteria.

Table 2
Usability as a software quality category

Category	Factor	Definition
Product usability	Learnability	The ease with which new users can begin effective interaction and achieve maximal performance.
	Interaction flexibility	Multiplicity of ways the user and system exchange information.
	Interaction robustness	Features of the interaction which support successful achievement and assessment of goals.

3. LEARNABILITY

Learnability concerns the features of the product that allow novice users to initially understand how to use it and then how to attain a maximal level of performance. Jordan *et al.* refer to the gap between the users initial performance capability and their experienced, maximal performance as the learnability gulf [12]. This includes not only the first time a user ever interacts with a particular system but also how the system supports the ability in the user to generalize from a few examples of interaction behavior to different and slightly varied examples. In this section, we identify criteria that contribute to learnability. The major criteria are discussed in separate subsections together with related criteria. The contents of this section are then summarized in Table 3 at the end of this section.

3.1. Predictability

Except when interacting with some video games, a user does not take very well to surprises. *Predictability* of an interactive system means that the users knowledge of the interaction history is sufficient to determine the result of future interaction. There are many degrees to which predictability can be satisfied. The knowledge can be restricted to the presently perceivable information, so that the user need not remember anything that is not currently observable. The knowledge requirement can be increased to the limit where the user is actually forced to remember what every previous keystroke was and what every previous screen display contained (and the order of each!) in order to fully determine the consequences of the next input action.

This notion of predictability is distinguished from deterministic behaviour of the computer system. Predictability is a user-centred concept; it is deterministic behaviour from the perspective of the user. Though a system-defined function, such as selection of a graphical object in a drawing package using a mouse, may be completely deterministic, for that function to be predictable the user must be aware of all of the information required to determine which object will be selected. So the user must be able to determine which set of objects the mouse is pointing to, what order those objects were created on the canvas, and what previous grouping commands had been performed to create compound objects from primitive objects. The user may even have to know which layer of the graphics package is the current one. If all of this information is not immediately perceivable by the user, then the user must remember some of the interaction history which is relevant to the selection algorithm. Predictability, once represented within some formalism, can then be used to provide cognitive requirements for the user, and these requirements will lead to a measure of the cognitive load that the system imposes on the user.

This notion of predictability deals with the users ability to determine the effect of operations on the system. Another form of predictability has to do with the users ability to know which operations can be performed. *Operation visibility* refers to the rendering of operations in a way that expresses their availability in the current state. If an operation can be performed, then there must be some perceivable indication of this to the user. Likewise, users should understand from the interface if an operation they might like to invoke cannot be performed. In addition, operations that cannot be performed should satisfy the do nothing principle [13]. Although perceivable, they cannot be invoked (e.g., dimmed menu items).

3.2. Synthesizability

Predictability focuses on the users ability to determine the effect of future interactions. This assumes that the user has some mental model of how the system behaves. Predictability says nothing about the way the user forms a model of the systems behaviour. In building up some sort of predictive model of the system's behavior, it is important for the user to assess the consequences of previous interactions in order to *synthesize* the behavior of the system. Synthesis, therefore, is the ability of the user to assess the effect of past operations on the

current state. We separate synthesis into two related activities, detection of change and detection of similarities.

When an operation changes some aspect of the internal state, it is important that that change be observable by the user. *Honesty* is the ability of the user interface to provide the user with an observable and informative account of such change. In the best of circumstances, this notification can come immediately, requiring no further interaction initiated by the user. Or at the very least, the notification should appear eventually after explicit user directives to make the change observable. The problem with eventual honesty is that the user must know to look for the change. In a situation where the user is learning a new interactive system, it is likely that he/she will not know to look for change. The eventual (accidental) discovery of a change may then be difficult to associate to a previous operation. It can appear eventually as a consequence of some user action which was not directly intended to expose the change (accidental discovery). In the worst case, some changes could be forever hidden from the user, in which case it would be impossible for the user to be able to associate such a change to its corresponding operation.

A user could be interested in detecting similarities between different commands. As we will discuss with interaction flexibility in Section 4, there is often more than one way to achieve a specified goal. In order to take advantage of this flexibility, the user must determine that two different command sequences yield the same result. This similarity can occur just after the execution of the different command sequences, in which case we would say the command sequences were *equivalent*. In the limit, the similarity may continue to any point in the future, in which case we would say the command sequences were *indistinguishable*. If the interactive system is immediately honest, equivalence command sequences would also be indistinguishable.

3.3. Familiarity

New users of a system bring with them a wealth of experience across a wide number of application domains. This experience is obtained both through interaction in the real world and also through interaction with previously existing computer systems. For a new user, the familiarity of an interactive system measures the correlation between the user's existing knowledge and the knowledge required for effective interaction. For example, when word processors were originally introduced, an analogy between the word processor and a typewriter were intended to make the new technology more immediately accessible to those who had little experience with the former and quite a bit of experience with the latter.

Familiarity has to do with a user's first impression of the system. In this case, we are interested in how the system is first perceived and whether the user can determine how to initiate any interaction. One of the advantages of a metaphor, such as the typewriter metaphor for word processing described above, is precisely captured by familiarity. Jordan *et al.* refer to this familiarity as the guessability gulf [12]. They have also shown how guessability can be directly measured based on a user's rate of completion of a set of tasks over time. In ecological psychology, one can refer to the *affordance* provided by perceivable interface objects [13, 14]. There is affordance when the perceivable intrinsic properties of any object instigate the appropriate actions upon it, that is, the appearance of the object stimulates a familiarity with its behavior. For example, the shape of a door handle can suggest how it should be manipulated to open a door, or a soft button used in a forms interface can suggest it should be pushed (though it does not suggest how it is to be pushed via the mouse). Taking advantage of the affordances which exist for interface objects enhances the predictability of an interface as well.

3.4. Generalizability

Users often try to extend their knowledge of specific interaction behavior to situations which are similar but previously unencountered. The generalizability of an interactive system supports this activity, leading to a more complete predictive model of the system for the user. We can apply generalization to situations in which the user wants to apply knowledge which

helped achieve a particular goal to another situation in which the goal is in some way similar. Generalizability can be seen as a form of consistency, and we can discuss its relevance within a single application and across a variety of applications. For example, using a graphical drawing package which draws a circle as a constrained form of ellipse, we would want the user to generalize that a square can be drawn as a constrained rectangle. A good example of generalizability across a variety of applications can be seen in multiwindowing systems which attempt to provide cut/paste/copy operations to all applications in the same way (with varying degrees of success).

3.5. Summary of Learnability

Table 3 summarizes the criteria we have discussed in this section which contribute to learnability.

Table 3
Summary of criteria affecting interaction learnability

Criterion	Definition	Related Criteria
Predictability	Support for the user to determine the affect of future action based on past interaction history.	Operation visibility, Consistency, Affordance
Synthesis	Support for the user to assess the affect of past operations on the current state.	Immediate/Eventual honesty, Equivalence, Indistinguishability
Familiarity	The extent to which a user's knowledge and experience in other real-world or computer-based domains can be applied when interacting with a new system.	Guessability, Affordance
Generalizability	Support for the user to extend knowledge of specific interaction within and across applications to other similar situations.	Consistency

4. INTERACTION FLEXIBILITY

Interaction flexibility refers to the multiplicity of ways the end user and the system exchange information. We identify several criteria that contribute to interaction flexibility, and these are summarized in Table 4 at the end of this section.

4.1. Dialogue initiative

When considering the interaction between user and system as a dialogue between partners, it is important to consider which partner has the initiative in the conversation. The system can initiate all dialogue, in which case the user simply responds to requests for information. We call this type of dialogue *system pre-emptive*. For example, a dialogue box may prohibit the user from interacting with the system in any way that does not direct input to the box. Alternatively, there are situations in which system pre-emptiveness can hinder a user's progress; it is desirable at times for the user to preempt the system. *User pre-emptiveness* defines the span for user freedom. The system may control the dialogue to the extent that it

prohibits the user from initiating any other desired communication pertaining to the same or different task [4]. From the user's perspective, a system-driven interaction hinders flexibility whereas a user-driven interaction favors it.

In general, we want to maximize the user's ability to pre-empt the system and minimize system's ability to preempt the user. Although non-pre-emptive systems are desirable, some situations may require it for safety reasons, prohibiting the user from the freedom to do potentially serious damage. In the extreme, this criterion would have the user being able to offer any input action at any time for maximum flexibility. This is not an entirely desirable situation, since it increases the likelihood that the user will lose track of the tasks that have been initiated and not yet completed. However, if the designer has a good understanding of the sets of tasks the user is likely to perform with a system and how those tasks are related, they can minimize the likelihood that the user will be prevented from initiating some task at a time when they would want.

4.2. Multithreading

Multithreading of the user/system dialogue allows for interaction to support more than one task at a time. *Concurrent multithreading* allows simultaneous communication of information pertaining to separate tasks. *Interleaved multithreading* permits a temporal overlap between separate tasks but stipulates that at any given instant, the dialogue is restricted to a single task. Multithreading contributes toward interaction flexibility since it allows the user to perform multiple tasks simultaneously or switch freely between them. There are many software engineering formalisms, most notably process algebras such as CSP [15] and CCS [16], which can describe multithreaded dialogues in which the threads are considered as independent sequences of events or actions. There is still a considerable research gap in software engineering in addressing how threads of events from the system perspective connect to the actual tasks in the work domain. Therefore, the value of process algebras as formalisms which lead to a measurement of multithreading capacity of the interaction is minimal at the moment.

4.3. Task migratability

Task migratability concerns the transfer of control for execution of tasks between system and user. It should be possible for the user or system to pass the control of a task over to the other or promote the task from a completely internalized one to a shared and co-operative venture. Hence, a task that is internal to one can become internal to the other or shared between the two partners. For example, OPAL, based on a model of operator intentions, is able to take over tasks normally assigned to an aircraft pilot [17].

4.4. Substitutivity

Substitutivity requires that equivalent values can be substituted for each other [3]. For example, the user may enter either 24 or 6*4 for some slot in a form. Entering 24 implies that the user performs the calculation. When the user submits 6*4, the system takes the responsibility for evaluating the expression. This input substitutivity contributes toward interaction flexibility by allowing the user to choose whichever form best suits the needs of the moment. By avoiding undesirable calculations, substitutivity can minimize user errors and cognitive effort.

We can also consider substitutivity with respect to output, or the system's rendering of state information. *Representation multiplicity* illustrates flexibility for state rendering. For example, the temperature of a physical object over a period of time can be presented as a digital thermometer if the actual numerical value is important or as a graph if it is important to notice trends. It might even be desirable to make these representations simultaneously available to the user. Each representation provides a perspective on the internal state of the system. At a given time, the user is free to consider the representations that are most suitable for the current task.

The PAC multiagent architecture [18] especially emphasizes this feature of multiple representation of a single system concept.

Equal opportunity blurs the distinction between input and output at the interface. The user has the choice of what is input and what is output; in addition, output can be reused as input [3]. If you can see it, you can use it! It is a common belief that input and output are separate. Many have stressed the significance of the link between input and output. Equal opportunity pushes that view to the extreme. For example, in spreadsheet programs, the user fills in some cells and the system automatically determines the values attributed to some other cells. Conversely, if the user enters values for those other cells, the system would compute the values for the first ones. In this example, it is not clear which cells are the inputs and which are the outputs. Furthermore, this distinction might not be clear nor useful to the user. In a drawing package, the user may draw a line by direct manipulation and the system would compute the length of the line; or conversely, the user may specify the line coordinates and the system would draw the line. Both means of manipulating the line are equally important and must be made equally available. Note that equal opportunity implies that the system is not pre-emptive toward the user. The spreadsheet example is just one type of constraints-based system which is very characteristic of this criterion. Formal notations are good for exploiting equal opportunity, for their abstractness can concentrate description on the constraints which exist between various system and interface attributes and not on their input/output characteristics.

4.5. Multimodality

Multimodality refers to the multiple use of communication channels[1]. Input from the user can originate from different channels, such as the keyboard, mouse, or voice. Output from the system can be received by the user through visual, audio, or haptic channels. Each different channel for the user is referred to as a modality of interaction. Multimodal systems may be characterized along two dimensions: fusion (exclusive or synergic) and time (sequentiality or concurrency) [19].

A user interface is *exclusive multimodal* if multiple modalities are available to the user, and input (and output) expressions are built up from one modality only (there is no fusion of modalities). For example, to open a window, the user can choose among a double-click on an icon, a keyboard shortcut, or say "open window". A system is *synergic multimodal* if input and output expressions can be expressed using a combination of modalities. For example, one can say “put that there” while pointing at the object to be moved and showing the location of the destination with the mouse or a data glove. In this formulation, speech events, such as “that” and “there”, call for complementary input events, such as mouse clicks and/or data glove events, in order to complete the input expression.

Time constraints express the possibility for the user to build exclusive or synergic expressions *sequentially* or in *parallel*. Sequentiality implies that there is no concurrency at the interface. For example, in the absence of concurrency, the user would say “put that” followed by a mouse click to denote “that”. He would then say “there” and click a second time to indicate the destination. Concurrency at the actions level supports “natural” flexibility for synergic multimodal interaction.

4.6. Customizability

Customizability is the modifiability of the user interface by the user or the system. From the system side, we are not concerned with modifications that would be attended to by a programmer actually changing the system and its interface. Rather, we are concerned with the automatic modification that the system would make based on its knowledge of the user. We

[1]There is a distinction between multimedia and multimodal user interfaces. Multimedia and multimodal systems use similar physical input and output devices. Both acquire, maintain and deliver visual and sonic information. Although similar at the surface level, they serve distinct purposes. Multimedia systems are concerned with the form whereas multimodal systems are concerned with meaning.

distinguish between the user-initiated and system-initiated modification, referring to the former as *adaptability* and the latter as *adaptivity*.

Adaptability refers to the user's ability to adjust the form of input and output. This customization could be very limited, with the user only allowed to adjust the position of soft buttons on the screen or redefine command names. This type of modifiability, which is restricted to the surface of the interface, is referred to as lexical and pragmatic customization. The overall structure of the interaction is kept unchanged. The power given to the user can be increased by allowing the definition of macros to speed up the articulation of certain common tasks. In the extreme, the interface can provide the user with programming language capabilities, such as the UNIX shell or the script language Hypertalk [20] in Hypercard. In these cases, Thimbleby points out that it would be suitable to invoke well-known principles of programming languages to the users interface programming language [4, 21].

Adaptivity is automatic customization of the user interface by the system. Decisions for adaptation can be based on user expertise or observed repetition of certain task sequences. The distinction between adaptivity and adaptability is that the user plays an explicit role in adaptability, whereas his role in an adaptive interface is more implicit. A system can be trained to recognize the behaviour of an expert or novice and accordingly adjust its dialogue control or help system automatically to match the needs of the current user. This is in contrast with a system which would require the user to explicitly classify themselves as novice or expert at the beginning of a session. Current research in the application of neural networks to pattern recognition problems in HCI addresses this type of adaptivity [22].

Automatic macro construction as proposed in Eager [23], combines adaptability with adaptivity in a simple and useful way. Repetitive tasks can be detected by observing user behaviour and macros can be automatically constructed from this observation to perform repetitive tasks automatically.

4.7. Summary of interaction flexibility

We summarize the criteria which contribute toward interaction flexibility in Table 4 below.

Table 4
Summary of criteria affecting interaction flexibility

Criterion	Definition	Related Criteria
Dialogue initiative	Allowing the user freedom from artificial constraints on the input dialogue imposed by the system.	System/User, pre-emptiveness
Multithreading	Ability of the system to support user interaction pertaining to more than one task at a time.	Concurrent vs. Interleaving
Task Migratability	The ability to pass control for the execution of a given task so that it becomes either internalized by the user or the system or shared between them.	
Substitutivity	Allowing equivalent values of input and output to be arbitrarily substituted for each other.	Representation multiplicity, Equal opportunity
Multimodality	The use of multiple human communication channels.	Exclusive/Synergic, Sequential/ Parallel
Customizability	Modifiability of the user interface by the user or the system.	Adaptivity, vs. Adaptability

5. INTERACTION ROBUSTNESS

A user is engaged with a computer in order to achieve some set of goals in the work or task domain. Interaction robustness covers features of this interaction which support the successful achievement and assessment of the goals. In the following subsections, we describe criteria which support interaction robustness. A summary of these criteria are presented in Table 5 at the end of the section.

5.1. Observability

Observability allows the user to evaluate the internal state of the system from the perceivable representation of that state. State evaluation allows the user to compare the current observed state with the state intended in the action plan, possibly leading to a plan revision. Observability is further divided into five criteria: browsability, default-ness, reachability, persistence and operation visibility. Operation visibility was covered in Section 3 with respect to predictability. The remaining four are discussed next.

Browsability allows the user to explore the current internal state of the system via the limited view provided at the interface. Usually the complexity of the domain does not allow the interface to show at once all of the relevant domain concepts. Indeed, this is one reason why the notion of task is used, in order to constrain the domain information needed at one time to a subset connected with the user's current activity. Even so, it is probable that all of the information a user needs to continue work is not immediately perceivable. There needs to be a way for the user to investigate, or browse, the internal state. And this browsing itself should not have any side-effects on that state, i.e., the browsing commands should be *passive* [24] with respect to the domain-specific parts of the internal state. Harrison and Dix have provided abstract and formal requirements for passive strategies used to observe all of the information contained in the internal state [25].

Default-ness assists the user by passive recall (i.e., a value is recognized as correct); it also reduces the number of physical actions necessary to specify a value. Thus, providing default values is a kind of error prevention mechanism. There are two kinds of default values: static and dynamic. *Static defaults* do not evolve with the session. They are either defined within the system or are acquired at initiation time from a profile file. On the other hand, *dynamic defaults* evolve during the session. They are computed by the system from previous user inputs.

Reachability refers to the possibility of navigation through the observable system states [3, 26]. There are various levels of reachability that have been given a formal definition, but the main notion is whether the user can navigate from any given state to any other state. Reachability in an interactive system affects the recoverability of the system, as we will discuss later. In addition, different levels of reachability can reflect the amount of interaction flexibility in the system as well, though we did not make that explicit in the discussion on flexibility.

Persistence deals with the duration of the effect of a communication act and the ability of the user to make use of that effect. The effect of vocal communication does not persist except in the memory of the receiver. Visual communication, on the other hand, can remain as an object which the user can subsequently manipulate long after the act of presentation.

5.2. Recoverability

Users make mistakes from which they want to recover. *Recoverability* is the ability to reach a desired goal after recognition of some error in previous interaction. There are two directions in which recovery can occur, forward or backward. Forward error recovery involves the acceptance of the current state and negotiation from that state toward the desired state. Backward error recovery is an attempt to undo the effects of previous interaction in order to resurrect a prior state from which to proceed toward the desired state.

Recovery can be initiated by the system or by the user. When performed by the system, recoverability is connected to the notion of fault-tolerance. This issue is one of the criteria

mentioned by McCall although fault-tolerance in software engineering is viewed from the functional perspective only. At the user interface level, the system can initiate a meta-dialogue to clarify user requests. For example, a speech recognizer may negotiate the meaning of an utterance. When performed by the user, it is important that a recoverability facility determine the intent of the user's recovery actions, i.e., whether they desire forward (negotiation) or backward (using undo/redo actions) corrective action. There are many formal definitions of backward error recovery mechanisms (see [3] for a clear review of some of these). Recoverability relies on reachability because we want to avoid blocking the user from getting to a desired state from some other undesired state (going down a blind alley).

In addition to providing the ability to recover, the procedure for recovery should reflect the work being done (or undone, as the case may be). *Commensurate effort* states that if it is difficult to undo a given effect on the state, then it should have been difficult to do in the first place. Conversely, easily undone actions should be easily doable. For example, if it is difficult to recover files which have been deleted in an operating system, then it should be difficult to remove them, or at least it should require more effort by the user to delete the file than to, say, rename it.

5.3. Response time

Response time measures the rate of communication between the system and the user. Response time is generally defined as the duration of time needed by the system to express state changes to the user. It depends on the computational resources involved to satisfy the user's request. In general, short durations and instantaneous response times are desirable. Instantaneous means that the user perceives system reactions as immediate. As significant are *response time conformance* and *response time stability*: Response time conformance expresses the adequacy of the duration as compared to user expectation. It enforces a feeling of good collaboration. Failure to satisfy response time conformance may result in an unusable system. Response time stability covers the invariance of the duration for identical or similar computational resources. For example, pull down menus are expected to pop up instantaneously as soon as a mouse button is pressed. Variations, that is failure to satisfy response time stability, would impede anticipation exploited by motor skill.

5.4. Task conformance

Since the purpose of an interactive system is to allow a user to perform various tasks in achieving certain goals within a specific application domain, we can ask whether the system supports all of the tasks of interest and whether it supports these as the user would want. *Completeness* addresses the coverage issue and *adequacy* addresses the user's understanding of the tasks.

It is not sufficient that the software product fully implements some set of computational services that were identified at early specification stages. It is essential that the system allows the user to achieve any of the desired tasks in a particular work domain as identified by a task analysis that precedes system specification. Further, it is not necessary that the system only support the tasks identified by a task analysis. Indeed, it is quite possible that the provision of a new computer-based tool will suggest to a user some tasks that were not even conceivable before the tool.

Task completeness is supported, for example, in the Diane method [27], which provides a framework and a notation for identifying and specifying the functions of the end user-computer system couple. A Diane analysis is task-oriented, making explicit the operations performed by the computer system and those performed by the final user. Similarly, MAD which embeds a method and a notation for task analysis [28], bridges the gap between the task analysis and the specification phase. User interface generators such as Sirocco [29] and UIDE [30] automatically produce a user interface from a high level conceptual description. This specification denotes the domain concepts and their relations as computational counterparts of the mental entities handled by the end user in the task domain. The automatic generation

provided by this type of tool guarantees the completeness of the user interface with regard to the specified domain-dependent concepts and functions, though it does not provide help for generalized tasks that might emerge with use of the system.

Discussion of task conformance has its roots in an attempt to formally model the meaning of direct manipulation by means of a conformance between a display and the underlying state it represents [25]. Abowd has since demonstrated the link between this conformance and standard data refinement in software engineering [31]. Completeness is only one aspect of this conformance/refinement relationship. With the intuition of the model-world metaphor [32], we demand that the task as represented by the world of the interface match the task as understood by the user and supported by the system. We use the term adequacy because the formal model resembles the adequacy condition in software engineering expressed between an abstract system specification and its set of possible concrete refinements.

5.5. Summary of interaction robustness

We summarize the criteria which contribute toward interaction robustness in Table 5.

Table 5
Summary of criteria affecting interaction robustness

Criterion	Definition	Related Criteria
Observability	Ability of the user to evaluate the internal state of the system from its perceivable representation.	Browsability, Static/Dynamic defaults, Reachability, Persistence, Operation visibility
Recoverability	Ability of the user to take corrective action once an error has been recognized.	Reachability, Forward vs. Backward recovery, Fault tolerance, Commensurate effort
Response time	How the user perceives the rate of communication with the system.	Conformance, Stability
Task conformance	The degree to which the system services support all of the tasks the user wishes to perform and in the way that the user understands them.	Completeness, Adequacy

6. CONCLUSIONS AND DIRECTIONS FOR RESEARCH

We have proposed 3 categories of quality factors, learnability, interaction flexibility and interaction robustness, which contribute to the usability of a software product. Within those categories we defined criteria which more directly relate to the interactive features of a software product. These can be used to clarify the notion of usability which has been relatively overlooked in the quality assurance work of software engineering. In so doing, we have addressed two of the four objectives for this research, namely, we have:

- defined an extensive, if not comprehensive, list of properties to guide the principled design of interactive systems;
- provided an organization for categorizing these properties which can easily be extended to account for other interactive properties.

We do not claim that the three categories of usability factors outlined in this paper represent a complete treatment of usability, which is why the second point above is important. In fact, we can see at least three categories which might be readily added.

From the ISO draft standard on usability measurement [33], we can highlight *user performance*, perhaps the ultimate measure for usability, at least in the context of that standard for office automation, which is only implicitly covered in some of the criteria mentioned above, and *user satisfaction*, which is not addressed at all.

In future extensions to this work, we anticipate that *consistency* will appear as a general category for the usability factor as well. We have mentioned consistency only briefly with respect to learnability. Such slight explicit mention of consistency is a serious problem, especially since this property is so widely acknowledged in just about every book on guidelines for user interface design. The reason for this is that we believe that consistency has many different forms, some of which appear in the above discussion of learnability. There is also considerable debate about the utility of consistency as a general guide for design [34].

The major emphasis in our work will now address how to use formalisms to represent these usability properties in a software engineering framework. We have cited throughout this paper several instances where this activity has already taken place.

The categorization of quality factors and criteria provided in this paper can also provide guidance to a design rationale, such as the Questions-Options-Criteria (QOC) notation [35]. QOC is an interesting illustration of traceability in user interface design. It makes explicit the questions a user interface designer asks during the design phase. The properties that we have catalogued in this paper can be used as the criteria to judge various design options for a design question.

Many now recognize the importance of integrating the results of HCI research with the practices of software engineering. We hope that this paper provides sufficient impetus in the direction of integration and serves as motivation for an interesting agenda for further research.

7. ACKNOWLEDGEMENTS

The authors would like to acknowledge the contributions from colleagues in both the ESPRIT BRA project 3066 (AMODEUS) and the IFIP Working Group 2.7 (Computer System User Interface). Many of our thoughts and words have been formed by stimulating conversations during the course of our research with these groups and we look forward to further discussions this work might inspire.

8. REFERENCES

1 V. R. Basili and J. D. Musa. The Future of Engineering Software: A Management Perspective. *IEEE Computer*, **24**(9), September, pp. 90-96, 1991.

2 Apple Computer Inc. *Human Interface Guidelines: The Apple Desktop Interface*. Addison-Wesley, 1987.

3 H. Thimbleby. *User Interface Design*; ACM Press, New York, Frontier Series, Addison-Wesley Publ., 1990.

4 H. Thimbleby. Design of interactive systems. In J. A. McDermid, editor, *The Software Engineers Reference Book*, chapter 57. Butterworths, 1991.

5 P. J. Barnard. Cognitive Resources and the Learning of Human-Computer Dialogs. In *Interfacing Thoughts, Cognitive Aspects of Human Computer Interaction*; J. Carroll ed., MIT Press, Cambridge, Mass., pp. 112-159, 1987.

6 R.M. Young, T.R.G. Green, T. Simon. Programmable User Models for Predictive Evaluation of Interface Design. *Proceedings of CHI89*. ACM, New York, pp.15-19, 1989.

7 J. Laird, A. Newell, and P. Rosenbloom. SOAR: an architecture for general intelligence. *Artificial Intelligence*, **33**:1-64, 1987.

8 J. McCall. *Factors in Software Quality*; General Electric Eds, 1977.

9 J.P. Cavano and J.A. McCall. A framework for the measurement of software quality. In *The Proceedings of the ACM Software Quality Assurance Workshop*, pp. 133-139. ACM, November 1978.

10 *IEEE Software Engineering Standards*; IEEE Press, 1987.

11 AFCIQ, Section logicielle, Groupe planification et coûts, S/Groupe Plan Assurance Qualité Logiciel PAQL. Recommandations de Plan dAssurance Qualité Logicielle; version V0 23/03/89, 1989.

12 P. W. Jordan, S. W. Draper, K. K. MacFarlane and S.-A. McNulty. Guessability, learnability, and experienced user performance. In D. Diaper and N. Hammond, editors, *People and Computers VI*. Proceedings of the HCI91 Conference. Cambridge University Press, pp. 237-245, 1991.

13 D.A. Norman.*Psychology of Everyday Things*, Basic Books Publi., 1988.

14 W. W. Gaver. Technology affordances. In *Reaching Through Technology: CHI91 Proceedings*. ACM Press, pp. 79-84, 1991.

15 C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall International, 1985.

16 R. Milner. *Communication and Concurrency*. Prentice Hall International, 1989.

17 B. Hardman Hoshstrasser and N. D. Geddes. OPAL, Operator Intent Inferencing for Intelligent Operator Support Systems; Technical Report, Search Technology, Inc., 4725 Peachtree Corners Circle, Norcross, GA 30092, 1989.

18 J. Coutaz. PAC, an Implementation Model for Dialog Design. In H.J. Bullinger and B. Shackel, editors, *Interact'87*, North-Holland, pp. 431-436, 1987.

19 J. Coutaz. Multimedia and Multimodal User Interfaces: A Taxonomy for Software Engineering Research Issues, *St Petersburg HCI Workshop*, August, 1992.

20 D. Shafer. *HyperTalk Programming*; MacIntosh Library, Hayden Books, Indianapolis, Indiana, 1988

21 R. D. Tenent. *Principles of Programming Languages*, Prentice-Hall, 1981.

22 R. Beale and J. Finlay, editors. *Neural Networks and Pattern Recognition in Human-Computer Interaction*. Ellis-Horwood, 1992.

23 A. Cypher. EAGER: Programming Repetitive Task by Example; *Proceedings of CHI91* (New Orleans, Apr. 28-May 2). ACM, New York, pp. 445-446, 1991.

24 A.J. Dix and M.D. Harrison: Formalising Models of Interaction in the Design of a Display Editor; In H. J. Bullinger and B. Shackel, editors, *Interact'87*, North-Holland, pp.409-414, 1987.

25 M. Harrison and A. Dix. A state model of direct manipulation in interactive systems. In M. Harrison and H. Thimbleby, editors, *Formal Methods in Human-Computer Interaction*, chapter 5. Cambridge University Press, 1990.

26 A.J. Dix. *Formal Methods for Interactive Systems*. Academic Press, 1991.

27 M.F. Barthet. *Logiciels interactifs et ergonomie, modèles et méthodes de conception*. Dunod informatique, 1989.

28 D. L. Scapin and C. Pierret-Golbriech. Towards a method for task description: MAD. In L. Berlinguet and D. Berthelette, editors, *Work with Display Units 89*. Elsevier Science Publishers. pp. 371-380, 1990.

29 V. Normand. Le modèle Sirocco: de la Spécification Conceptuelle des Interfaces Utilisateur à leur réalisation. Thèse de doctorat, Université Joseph Fourier, to appear, 1992.

30 J.D. Foley, W.C. Kim, S. Kovacevic, K. Murray. The User Interface Design Environment, Technical Report GWU-IIST-88-04, Department of Electrical Engineering and Computer Science, The George Washington University, Washington D.C. 20052, January, 1988.

31 G. D. Abowd.Using formal methods for the specification of user interfaces. In R. Selby, editor, *Proceedings of the 2nd Annual Irvine Software Symposium, ISS92*. pp. 109-130. March, 1992.

32 E. L. Hutchins, J. D. Hollan and D. A. Norman. Direct Manipulation Interfaces. In D. A. Norman and S. W. Draper, editors, *User-Centered System Design*, pp. 87-124. Erlbaum, 1986.

33 ISO. Ergonomic requirements for office work with visual display terminals. Part 11: Guidance on usability specification and measures. ISO Committee Draft standard ISO-CD-9241-11.2, 1992.

34 J. Grudin. The Case Against User Interface Consistency; *Communications of the ACM*, 32(10), pp. 1164-1173, 1989.

35 A. MacLean, R. Young, V. Bellotti, T. Moran. Design Space Analysis: Bridging from Theory to Practice via Design Rationale. *ESPRIT91* conference proceedings. Commission of the European Communities, DG XIII, pp. 720-730, 1991.