CHAPTER VI

6 Design rules

6. Design Rules

- Designing for maximum usability is the goal of interactive systems design.
- Abstract principles offer a way of understanding usability in a more general sense, especially if we can express them within some coherent catalog.
- Design rules in the form of standards and guidelines provide direction for design, in both general and more concrete terms, in order to enhance the interactive properties of the system.
- The essential characteristics of good design are often summarized through ‘golden rules’ or heuristics.
- Design patterns provide a potentially generative approach to capturing and reusing design knowledge.

6.1 Introduction

One of the central problems that must be solved in a user-centered design process is how to provide designers with the ability to determine the usability consequences of their design decisions. The *design rules*, which are rules a designer can follow in order to increase the usability of the eventual software product.

It can classify these rules along two dimensions, based on the rule’s authority and generality.

By authority, we mean an indication of whether or not the rule must be followed in design or whether it is only suggested.

By generality, we mean whether the rule can be applied to many design situations or whether it is focussed on a more limited application situation. Rules also vary in their level of abstraction, with some abstracting away from the detail of the design solution and others being quite specific.

We will consider a number of different types of design rules.

Principles are abstract design rules, with high generality and low authority.

Standards are specific design rules, high in authority and limited in application, whereas *guidelines* tend to be lower in authority and more general in application.

Design rules for interactive systems can be supported by psychological, cognitive, ergonomic, sociological, economic or computational theory, which may or may not have roots in empirical evidence.

The design rules are used to apply the theory in practice. Often a set of design rules will be in conflict with each other, meaning that strict adherence to all of them is impossible.

The theory underlying the separate design rules can help the designer understand the trade-off for the design that would result in following or disregarding some of the rules.

6.2 Principles To Support Usability

The principles we present are first divided into three main categories:

- **Learnability** – the ease with which new users can begin effective interaction and achieve maximal performance.
- **Flexibility** – the multiplicity of ways in which the user and system exchange information.
- **Robustness** – the level of support provided to the user in determining successful achievement and assessment of goals.

6.2.1 Learnability

Learnability concerns the features of the interactive system that allow novice users to understand how to use it initially and then how to attain a maximal level of performance.

Predictability of an interactive system means that the user's knowledge of the interaction history is sufficient to determine the result of his future interaction with it.

Predictability of an interactive system is distinguished from deterministic behavior of the computer system alone.

Operation visibility

It refers to how the user is shown the availability of operations that can be performed next. If an operation can be performed, then there may be some perceivable indication of this to the user.

Synthesizability

Predictability focuses on the user's ability to determine the effect of future interactions.

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 through interaction with other 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.

Generalizability

Users often try to extend their knowledge of specific interaction behavior to situations that 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.

Consistency

Consistency relates to the likeness in behavior arising from similar situations or similar task objectives.

6.2.2 Flexibility

Flexibility refers to the multiplicity of ways in which the end-user and the system exchange information.

- *Dialog initiative*
- *Multi-threading*
- *Task migratability*
- *Substitutivity*
- *Equal opportunity*
- *Customizability*
- Adaptability
- Adaptivity

6.2.3 Robustness

The robustness of that interaction covers features that support the successful achievement and assessment of the goals.

- Observability
- Recoverability
- Responsiveness
- Task conformance
- Related principles
- Browsability, static/dynamic defaults, reachability, persistence, operation visibility
- Reachability, forward/backward recovery, commensurate effort
- Stability
- Task completeness, task adequacy

6.3 Standards

Standards for interactive system design are usually set by national or international bodies to ensure compliance with a set of design rules by a large community. Standards can apply specifically to either the hardware or the software used to build the interactive system.

Underlying theory Standards for hardware are based on an understanding of physiology or ergonomics/human factors, the results of which are relatively well known, fixed and readily adaptable to design of the hardware.

Change Hardware is more difficult and expensive to change than software, which is usually designed to be very flexible. Consequently, requirements changes for hardware do not occur as frequently as for software.

Since standards are also relatively stable, they are more suitable for hardware than software. Historically, for these reasons, a given standards institution, such as the British Standards Institution (BSI) or the International Organization for Standardization (ISO)

One component of the ISO standard 9241, pertaining to usability specification, applies equally to both hardware and software design. In the beginning of that document, the following definition of usability is given:

Usability The effectiveness, efficiency and satisfaction with which specified users achieve specified goals in particular environments.

Effectiveness The accuracy and completeness with which specified users can achieve specified goals in particular environments.

Efficiency The resources expended in relation to the accuracy and completeness of goals achieved.

Satisfaction The comfort and acceptability of the work system to its users and other people affected by its use.

6.4 Guidelines

We have observed that the incompleteness of theories underlying the design of interactive software makes it difficult to produce authoritative and specific standards. As a result, the majority of design rules for interactive systems are suggestive and more general guidelines.

The basic categories of the Smith and Mosier guidelines are:

1. Data Entry
2. Data Display
3. Sequence Control
4. User Guidance
5. Data Transmission

6. Data Protection

Each of these categories is further broken down into more specific subcategories which contain the particular guidelines.

This guideline supports several of the principles discussed in this chapter:

Substitutivity The system is able to substitute color-coded information and other means (for example, text, sound) to represent some important information.

Observability This principle is all about the system being able to provide the user with enough information about its internal state to assist his task. Relying strictly on color-coded information, as pointed out above, could reduce the observability of a system for some users.

Synthesis If a change in color is used to indicate the changing status of some system entity.

6.5 Implementation Support

- Programming tools for interactive systems provide a means of effectively translating abstract designs and usability principles into an executable form. These tools provide different levels of services for the programmer.
- Windowing systems are a central environment for both the programmer and user of an interactive system, allowing a single workstation to support separate user–system threads of action simultaneously.
- Interaction toolkits abstract away from the physical separation of input and output devices, allowing the programmer to describe behaviors of objects at a level similar to how the user perceives them.
- User interface management systems are the final level of programming support tools, allowing the designer and programmer to control the relationship between the presentation objects of a toolkit with their functional semantics in the actual application.

6.6 Elements of Windowing Systems

The first important feature of a windowing system is its ability to provide programmer independence from the specifics of the hardware devices. A typical workstation will involve some visual display screen, a keyboard and some pointing device, such as a mouse. Any variety of these hardware devices can be used in any interactive system and they are all different in terms of the data they communicate and the commands that are used to instruct them. It is imperative to be able to program an application that will run on a wide range of devices.

Examples of imaging models

Pixels
The display screen is represented as a series of columns and rows of points – or pixels – which can be explicitly turned on or off, or given a color. This is a common imaging model for personal computers and is also used by the X windowing system.

Graphical kernel system (GKS)
An international standard which models the screen as a collection of connected segments, each of which is a macro of elementary graphics commands.

Programmer's hierarchical interface to graphics (PHIGS)
Another international standard, based on GKS but with an extension to model the screen as editable segments.

PostScript
A programming language developed by Adobe Corporation which models the screen as a collection of paths which serve as infinitely thin boundaries or stencils which can be filled in with various colors or textured patterns and images.

The X server performs the following tasks:

- allows (or denies) access to the display from multiple client applications;
- interprets requests from clients to perform screen operations or provide other information;
- demultiplexes the stream of physical input events from the user and passes them to the appropriate client;
- minimizes the traffic along the network by relieving the clients from having to keep track of certain display information, like fonts, in complex data structures that the clients can access by ID numbers.

6.7 Programming the Application

We now concentrate our attention on programming the actual interactive application, which would correspond to a client in the client–server architecture. Interactive applications are generally user driven in the sense that the action the application takes is determined by the input received from the user.

We describe two programming paradigms which can be used to organize the flow of control with in the application.

The windowing system does not necessarily determine which of these two paradigms is to be followed.

The first programming paradigm is the *read–evaluation loop*, which is internal to the application program itself. Programming on the Macintosh follows this paradigm. The server sends user inputs as structured events to the client application. As far as the server is concerned, the only importance of the event is the client to which it must be directed.

The other programming paradigm is *notification based*, in which the main control loop for the event processing does not reside within the application. Instead, a centralized *notifier* receives events from the window system and filters them to the application program in a way declared by the program.

After processing, the callback procedure returns control to the notifier, either telling it to continue receiving events or requesting termination.

6.8 Using Toolkits

This creates the illusion that the entities on the screen are the objects of interest – interaction objects we have called them – and that is necessary for the action world of a direct manipulation interface.

A classic example is the mouse as a pointing device. The input coming from the hardware device is separate from the output of the mouse cursor on the display screen. However, since the visual movement of the screen cursor is linked with the physical movement of the mouse device, the user feels as if he is actually moving the visual cursor.

To aid the programmer in fusing input and output behaviors, another level of abstraction is placed on top of the window system – the *toolkit*. A toolkit provides the programmer with a set of ready-made interaction objects – alternatively called interaction techniques, gadgets or widgets – which she can use to create her application programs.

Two features of interaction objects and toolkits make them amenable to an object oriented approach to programming.

First, they depend on being able to define a class of interaction objects which can then be invoked (or instantiated) many times with in one application with only minor modifications to each instance.

Secondly, building complex interaction objects is made easier by building up their definition based on existing simpler interaction objects. These notions of *instantiation* and *inheritance* are cornerstones of object-oriented programming. *Classes* are defined as templates for interaction objects.

6.9 User Interface Management Systems

Toolkits provide only a limited range of interaction objects, limiting the kinds of interactive behavior allowed between user and system. Toolkits are expensive to create and are still very difficult to use by non-programmers.

Even experienced programmers will have difficulty using them to produce an interface that is predictably usable. There is a need for additional support for programmers in the design and use of toolkits to overcome their deficiencies.

The set of programming and design techniques which are supposed to add another level of services for interactive system design beyond the toolkit level are *user interface management systems*, or *UIMS* for short.

The main concerns of a UIMS, for our purposes, are:

- a conceptual architecture for the structure of an interactive system which concentrates on a separation between application semantics and presentation;
- techniques for implementing a separated application and presentation whilst preserving the intended connection between them;
- support techniques for managing, implementing and evaluating a run-time interactive environment.

6.9.1 UIMS as a Conceptual Architecture

A major issue in this area of research is one of *separation* between the semantics of the application and the interface provided for the user to make use of that semantics.

There are many good arguments to support this separation of concerns:

Portability To allow the same application to be used on different systems it is best to consider its development separate from its device-dependent interface.

Reusability Separation increases the likelihood that components can be reused in order to cut development costs.

Multiple interfaces To enhance the interactive flexibility of an application, several different interfaces can be developed to access the same functionality.

Customization The user interface can be customized by both the designer and the user to increase its effectiveness without having to alter the underlying application.

Once we allow for a separation between application and presentation, we must consider how those two partners communicate. This role of communication is referred to as *dialog control*.

Conceptually, this provides us with the three major components of an interactive system: the application, the presentation and the dialog control.

The logical components of a UIMS were identified as:

Presentation The component responsible for the appearance of the interface, including what output and input is available to the user.

Dialog control The component which regulates the communication between the presentation and the application.

Application interface The view of the application semantics that is provided as the interface.

Another so-called *multi-agent* architecture for interactive systems is the *presentation–abstraction–control* PAC model suggested by Coutaz.

PAC is based on a collection of triads also: with application semantics represented by the abstraction component; input and output combined in one presentation component; and an explicit control component to manage the dialog and correspondence between application and presentation.

6.9.2 Implementation Considerations

Window systems and toolkits provide the separation between application and presentation. The use of callback procedures in notification-based programming is one way to implement the application interface as a notifier.

In the standard X toolkit, these callbacks are directional as it is the duty of the application to register itself with the notifier.

In MVC, callback procedures are also used for communication between a view or controller and its associated model, but this time it is the duty of the presentation (the view or controller)to register itself with the application (the model).

Some of the techniques that have been used in dialog modeling in UIMS are listed here.

Menu networks The communication between application and presentation is modeled as a network of menus and submenus. Links between menu items and the next displayed menu model the application response to previous input. A menu does not have to be a linear list of textual actions. The menu can be represented as graphical items or buttons that the user can select with a pointing device.

Grammar notations The dialog between application and presentation can be treated as a grammar of actions and responses, and, therefore, described by means of a formal context-free grammar notation.

State transition diagrams State transition diagrams can be used as a graphical means of expressing dialog. The difficulty with these notations lies in linking dialog events with corresponding presentation or application events. Also, it is not clear how communication between application and presentation is represented.

Event languages Event languages are similar to grammar notations, except that they can be modified to express directionality and support some semantic feedback. Event languages are good for describing localized input–output behavior in terms of production rules. A production rule is activated when input is received and it results in some output responses.

Declarative languages A declarative approach concentrates more on describing how presentation and application are related. This relationship can be modeled as a shared database of values that both presentation and application can access.

Constraints systems are a special subset of declarative languages.

Constraints can be used to make explicit the connection between independent information of the presentation and the application.

Graphical specification These techniques allow the dialog specification to be programmed graphically in terms of the presentation language itself. This technique can be referred to as *programming by demonstration* since the programmer is building up the interaction dialog directly in terms of the actual graphical interaction objects that the user will see, instead of indirectly by means of some textual specification language that must still be linked with the presentation objects.