**DESIGN & SOFTWARE PROCESS**

Interactive Design basics - process - scenarios - navigation - screen design - Iteration and prototyping. HCI in software process - software life cycle - usability engineering - Prototyping in practice - design rationale. Design rules - principles, standards, guidelines, rules. Evaluation Techniques - Universal Design.

**2.1 INTERACTION DESIGN BASICS**

Interaction design is about creating interventions in often complex situations using technology of many kinds including PC software, the web and physical devices.

• Design involves:

* + 1. Achieving goals within constraints and trade-off between these
    2. Understanding the raw materials: computer and human 3 Accepting limitations of humans and of design.
  + The design process has several stages and is iterative and never complete.
  + Interaction starts with getting to know the users and their context:

1. Finding out who they are and what they are like . . .probably not like you!

2 Talking to them, watching them.

* + Scenarios are rich design stories, which can be used and reused throughout design: they help us see what users will want to do

they give a step-by-step walkthrough of users‗ interactions: including what they see, do and are thinking.

* + Users need to find their way around a system. This involves:
    1. Helping users know where they are, where they have been and what they can do next
    2. Creating overall structures that are easy to understand and fit the users‗ needs 3 Designing comprehensible screens and control panels.

• Complexity of design means we don‗t get it right first time:

* + 1. So we need iteration and prototypes to try out and evaluate
    2. But iteration can get trapped in local maxima; designs that have no simple improvements, but are not good theory and models can help give good start points.

**WHAT IS DESIGN?**

A simple definition is: achieving goals within constraints

**Goals:** what is the purpose of the design we are intending to produce? Who is it for? Why do they want it? For example, if we are designing a wireless personal movie player, we may think about young affluent users wanting to watch the latest movies whilst on the move and download free copies, and perhaps wanting to share the experience with a few friends.

**Constraints:** What materials must we use? What standards must we adopt? How much can it cost? How much time do we have to develop it? Are there health and safety issues? In the case of the personal movie player: does it have to withstand rain? Must we use existing video standards to download movies? Do we need to build in copyright protection?

**Trade-off** Choosing which goals or constraints can be relaxed so that others can be met. For example, we might find that an eye-mounted video display, a bit like those used in virtual reality, would give the most stable image whilst walking along. However, this would not allow you to show friends, and might be dangerous if you were watching a gripping part of the movie as you crossed the road.

**The golden rule of design**

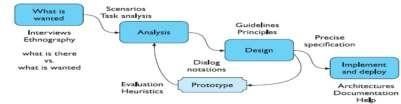
The designs we produce may be different, but often the raw materials are the same. This leads us to the golden rule of design: **understand your materials**

* understand computers o limitations, capacities, tools, platforms
* understand people o psychological, social aspects, human error.

**2.2 PROCESS**

**2.2.1 The Process of Design**

A system has been designed and built, and only when it proves unusable do they think to ask how to do it right! In other companies usability is seen as equivalent to testing – checking whether people can use it and fixing problems, rather than making sure they can from the beginning. In the best companies, however, usability is designed in from the start.



**Figure: Interaction design process**

**Requirements –** what is wanted The first stage is establishing what exactly is needed. As a precursor to this it is usually necessary to find out what is currently happening.

**Analysis:** The results of observation and interview need to be ordered in some way to bring out key issues and communicate with later stages of design.

**Design**: Well, this is all about design, but there is a central stage when you move from what you want, to how to do it. There are numerous rules, guidelines and design principles that can be used to help

**Iteration and prototyping:** Humans are complex and we cannot expect to get designs right first time. We therefore need to evaluate a design to see how well it is working and where there can be improvements.

**Implementation and deployment** Finally, when we are happy with our design, we need to create it and deploy it. This will involve writing code, perhaps making hardware, writing documentation and manuals – everything that goes into a real system that can be given to others.

**2.3 Scenarios**

Scenarios are stories for design: rich stories of interaction. They are perhaps the simplest design representation, but one of the most flexible and powerful. Some scenarios are quite short: ‗the user intends to press the ―save‖ button, but accidentally presses the ―quit‖ button so loses his work‗. Others are focussed more on describing the situation or context.

Scenarios force you to think about the design in detail and notice potential problems before they happen. What is the system doing now?‗. This can help to verify that the design would make sense to the user and also that proposed implementation architectures would work.

**In addition scenarios can be used to:**

**Communicate with others** – other designers, clients or users. It is easy to misunderstand each other whilst discussing abstract ideas. Concrete examples of use are far easier to share.

**Validate other models**: A detailed scenario can be ‗played‗ against various more formal representations such as task models or dialog and navigation models .

**Express dynamics** Individual screen shots and pictures give you a sense of what a system would look like, but not how it behaves.

**2.4 Navigation Design**

Navigation Design is the process or activity of accurately ascertaining

one's position and planning and following a route. the process or activity of accurately ascertaining one's position and planning and following a route.

**Widgets** The appropriate choice of widgets and wording in menus and buttons will help you know how to use them for a particular selection or action.

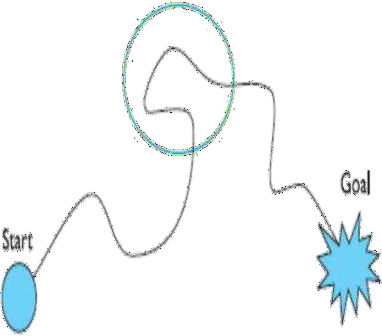
**Screens or windows** You need to find things on the screen, understand the logical grouping of buttons.

**Navigation within the application** You need to be able to understand what will happen when a button is pressed, to understand where you are in the interaction.

**Environment** The word processor has to read documents from disk, perhaps some are on remote networks. You swap between applications, perhaps cut and paste.

**2.4.1 Local structure**

In an ideal world if users had perfect knowledge of what they wanted and how the system worked they could simply take the shortest path to what they want, pressing all the right buttons and links. The important thing is not so much that they take the most efficient route, but that at each point in the interaction they can make some assessment of whether they are getting closer to their (often partially formed) goal.

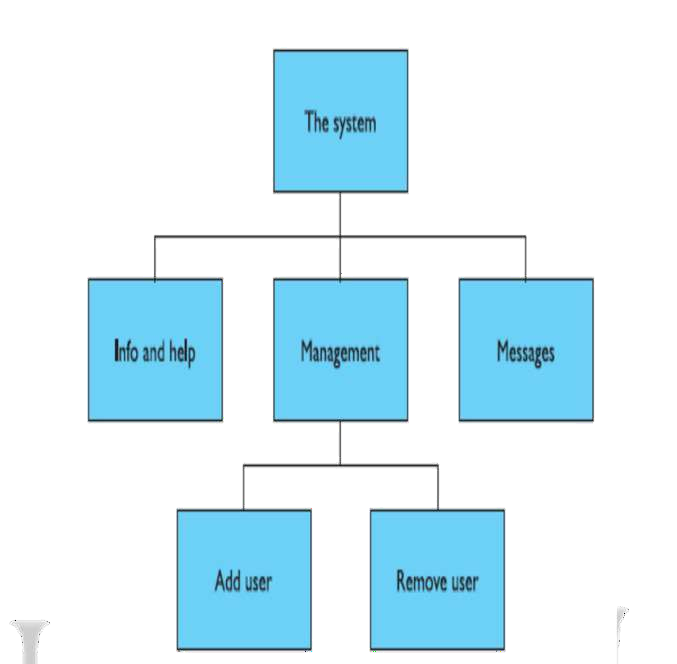
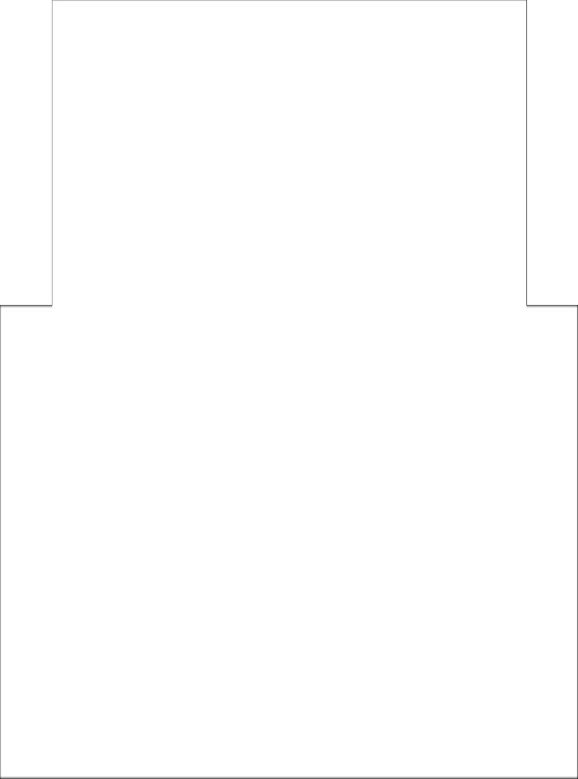


To do this goal seeking, each state of the system or each screen needs to give the user enough knowledge of what to do to get closer to their goal.

* knowing where you are
* knowing what you can do
* knowing where you are going – or what will happen
* knowing where you‗ve been – or what you‗ve done.

**2.4.2 Global structure – hierarchical organization**

The hierarchy links screens, pages or states in logical groupings. The Diagram gives a high-level breakdown of some sort of messaging system. This sort of hierarchy can be used purely to help during design, but can also be used to structure the actual system. For example, this may reflect the menu structure of a PC application or the site structure on the web. Any sort of information structuring is difficult, but there is evidence that people find hierarchies simpler than most. One of the difficulties with organizing information or system functionality is that different people have different internal structures for their knowledge, and may use different vocabulary.



**Figure:** Application functional hierarchy

**2.3 SCREEN DESIGN AND LAYOUT**

**2.3.1 Tools for layout**

We have a number of visual tools available to help us suggest to the user appropriate ways to read and interact with a screen or device.



Figure: Grouping related items in an order screen

**Grouping and structure**

If things logically belong together, then we should normally physically group them together. This may involve multiple levels of structure. We can see a potential design for an ordering screen. Notice how the details for billing and delivery are grouped together spatially; also note how they are separated from the list of items actually ordered by a line as well as spatially. This reflects the following logical structure:

**Order:**

* Administrative information o Billing details o Delivery details
* Order information o Order line 1 o Order line 2

**Order of groups and items**

In general we need to think: what is the natural order for the user? This should normally match the order on screen. For data entry forms or dialog boxes we should also set up the order in which the tab key moves between fields. Occasionally we may also want to force a particular order; for example we may want to be sure that we do not forget the credit card details

**Decoration**

Decorative features like font style, and text or background colors can be used to emphasize groupings.

**Alignment**

Alignment of lists is also very important. For users who read text from left to right, lists of text items should normally be aligned to the left. Numbers, however, should normally be aligned to the right (for integers) or at the decimal point. This is because the shape of the column then gives an indication of magnitude – a sort of mini histogram. Items like names are particularly difficult.

**White space**

Spacing or whitespace, white space is any section of a document that is unused or space around an object. White spaces help separate paragraphs of text, graphics, and other portions of a document, and help a document look less crowded. Using white space effectively in a document keeps the reader reading the document and helps the reader quickly find what they are interested in reading.

**How to create white space**

White space is created by pressing the return key, spacebar key, or the

tab key and can also be created by setting the document's margins and inserting form feeds or tables.

**2.3.2 User action and control**

• Entering information

In each case the screen consists not only of information presented to the user, but also of places for the user to enter information or select options. Many of the same layout issues for

data presentation also apply to fields for data entry. Alignment is still important. It is

especially common to see the text entry boxes aligned in a jagged fashion because the field names are of different lengths. This is an occasion where right-justified text for the field labels may be best or, alternatively, in a graphical interface a smaller font can be used for field labels and the labels placed just above and to the left of the field they refer to. For both presenting and entering information a clear logical layout is important.

The task analysis techniques can help in determining how to group screen items and also the order in which users are likely to want to read them or fill them in. Knowing also that users are likely to read from left to right and top to bottom (depending on their native language!) means that a screen can be designed so that users encounter items in an appropriate order for the task at hand.

* **Knowing what to do**

If everyone designs buttons to look the same and menus to look the same,

then users will be able to recognize them when they see them. It is important that the labels and icons on menus are also clear. Standards can help for common actions such as save, delete or print. For more system-specific actions, one needs to follow broader principles. For example, a button says ‗bold‗: does this represent the current state of a system or the action that will be performed if the button is pressed?

* **Affordances**

These are especially difficult problems in multimedia applications where one

may deliberately adopt a non-standard and avant-garde style. How are users supposed to know where to click? The psychological idea of affordance says that things may suggest by their shape and other attributes what you can do to them: a handle affords pulling or lifting; a button affords pushing. These affordances can be used when designing novel interaction elements. One can either mimic real-world objects directly, or try to emulate the critical aspects of those objects. What you must not do is depict a real-world object in a context where its normal affordances do not work!

**2.3.3 Appropriate appearance**

* **Presenting information**

The way of presenting information on screen depends on the kind of information: text, numbers, maps, tables; on the technology available to present it: character display, line drawing, graphics, and virtual reality; and, most important of all, on the purpose for which it is being used. The file listing is alphabetic, which is fine if we want to look up the details of a particular file, but makes it very difficult to find recently updated files. Of course, if the list were ordered by date then it would be difficult to find a particular file. Different purposes require different representations. For more complex numerical data, we may be considering scatter graphs, histograms or 3D surfaces; for hierarchical structures, we may consider outlines or organization diagrams. But, no matter how complex the data, the principle of matching presentation to purpose remains. We have an advantage when presenting information in an interactive system in that it is easy to allow the user to choose among several representations, thus making it possible to achieve different goals.



Figure : Alphabetic file listing. Screen shot reprinted by permission from Apple Computer, Inc.

**Aesthetics and utility**

The beauty and utility may sometimes be at odds. For example, an industrial control panel will often be built up of the individual controls of several subsystems, some designed by different teams, some bought in. The resulting inconsistency in appearance may look a mess and suggest tidying up. Certainly some of this inconsistency may cause problems. The conflict between aesthetics and utility can also be seen in many

‗well designed‗ posters and multimedia systems. In particular, the backdrop behind text must have low contrast in order to leave the text readable; this is often not the case and graphic designers may include excessively complex and strong backgrounds because they look good. The results are impressive, perhaps even award winning, but completely unusable! In consumer devices these aesthetic considerations may often be the key differentiator betweenproducts, for example, the sleek curves of a car. This is not missed by designers of electronic goods: devices are designed to be good to touch and feel as well as look at and this is certainly one of the drivers for the futuristic shapes of the Apple iMac family.

**Making a mess of it: colour and 3D**

The increasing use of 3D effects in interfaces has posed a whole new set of problems for text and numerical information. Whilst excellent for presenting physical information and certain sorts of graphs, text presented in perspective can be very difficult to read and the all too common 3D pie chart is all but useless.

**Localization / internationalization**

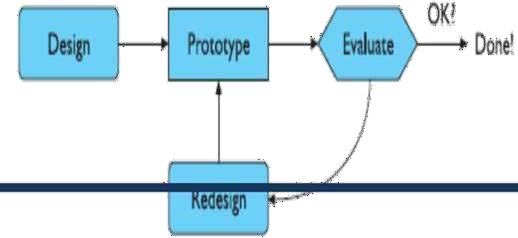
If you are working in a different country, you might see a document being word processed where the text of the document and the file names are in the local language, but all the menus and instructions are still in English. The process of making software suitable for different languages and cultures is called localization or internationalization.

It is clear that words have to change and many interface construction toolkits make this easy by using resources. When the program uses names of menu items, error messages and other text, it does not use the text directly, but instead uses a resource identifier, usually simply a number. A simple database is constructed separately that binds these identifiers to particular words and phrases. A different resource database is constructed for each language, and so the program can be customized to use in a particular country by simply choosing the appropriate resource database.

**2.4 ITERATION AND PROTOTYPING**

All interaction design includes some form of iteration of ideas. This often starts early on with paper designs and storyboards being demonstrated to colleagues and potential users. Any of these prototypes, whether paper-based or running software, can then be evaluated to see whether they are acceptable and where there is room for improvement. This sort of evaluation, intended to improve designs, is called formative evaluation. This is in contrast to summative evaluation, which is performed at the end to verify whether the product is good enough. One approach is to get an expert to use a set of guidelines, for example the ‗knowing where you are‗ list above, and look screen by screen to see if there are any violations.

The other main approach is to involve real users either in a controlled experimental setting, or ‗in the wild‗ – a real-use environment. The result of evaluating the system will usually be a list of faults or problems and this is followed by a redesign exercise, which is then prototyped, evaluated The end point is when there are no more problems that can economically be fixed. So iteration and prototyping are the universally accepted ‗best practice ‗approach for interaction design.



**Figure :Role of prototyping**

Prototyping is an example of what is known as a hill-climbing approach. Imagine you are standing somewhere in the open countryside. You walk uphill and keep going uphill as

steeply as possible. Eventually you will find yourself at a hill top.is

exactly how iterative prototyping works: you start somewhere, evaluate it to see how to make

it better, change it to make it better and then keep on doing this until it can‗t get any better.

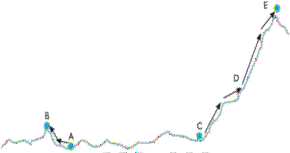


Figure: Moving little by little ....... but to where?

**2.5 HCI IN THE SOFTWARE PROCESS**

* Software engineering provides a means of understanding the structure of the design process, and that process can be assessed for its effectiveness in interactive system design.
* Usability engineering promotes the use of explicit criteria to judge the success of a product in terms of its usability.
* Iterative design practices work to incorporate crucial customer feedback early in the design process to inform critical decisions which affect usability.
* Design involves making many decisions among numerous alternatives. Design rationale provides an explicit means of recording those design decisions and the context in which the decisions were made.

**2.6 Software Life cycle models**

In the development of a software product, we consider two main parties: the customer who requires the use of the product and the designer who must provide the product. Typically, the customer and the designer are groups of people and some people can be both customer and designer. It is often important to distinguish between the customer who is the client of the designing company and the customer who is the eventual user of the system. These two roles of customer can be played by different people. The group of people who negotiate the features of the intended system with the designer may never be actual users of the system. This is often particularly true of web applications. In this chapter, we will use the term ‗customer‗ to refer to the group of people who interact with the design team and we will refer to those who will interact with the designed system as the user or end-user.

**2.6.1 Activities**

The graphical representation is reminiscent of a waterfall, in which each activity naturally leads into the next. The analogy of the waterfall is not completely faithful to the real relationship between these activities, but it provides a good starting point for discussing the logical flow of activity. We describe the activities of this waterfall model of the software life cycle

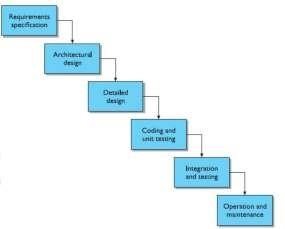


Figure The activities in the waterfall model of the software life cycle

**Requirements specification**

Requirements specification begins at the start of product development. Though the requirements are from the customer‗s perspective, if they are to be met by the software product they must be formulated in a language suitable for implementation. Requirements are usually initially expressed in the native language of the customer. The executable languages for software are less natural and are more closely related to a mathematical language in which each term in the language has a precise interpretation, or semantics. The transformation from the expressive but relatively ambiguous natural language of requirements to the more precise but less expressive executable languages is one key to successful development. Task analysis techniques, which are used to express work domain requirements in a form that is both expressive and precise.

**Architectural design**

The requirements specification concentrates on what the system is supposed to do. The next activities concentrate on how the system provides the services expected from it. The first activity is a high-level decomposition of the system into components that can either be brought in from existing software products or be developed from scratch independently. An architectural design performs this decomposition. It is not only concerned with the functional decomposition of the system, determining which components provide which services. It must also describe the interdependencies between separate components and the sharing of resources that will arise between components.

**Detailed design**

The architectural design provides a decomposition of the system description that allows for isolated development of separate components which will later be integrated. For those components that are not already available for immediate integration, the designer must provide a sufficiently detailed description so that they may be implemented in some programming language. The detailed design is a refinement of the component description provided by the architectural design. The behavior implied by the higher-level description must be preserved in the more detailed description.

There will be more than one possible refinement of the architectural component that will satisfy the behavioral constraints. Choosing the best refinement is often a matter of trying to satisfy as many of the non-functional requirements of the system as possible. Thus the language used for the detailed design must allow some analysis of the design in order to assess its properties.

**Coding and unit testing**

The detailed design for a component of the system should be in such a form that it is possible to implement it in some executable programming language. After coding, the component can be tested to verify that it performs correctly, according to some test criteria that were determined in earlier activities. Research on this activity within the life cycle has concentrated on two areas. There is plenty of research that is geared towards the automation of this coding activity directly from a low-level detailed design. Most of the work in formal methods operates under the hypothesis that, in theory, the transformation from the detailed design to the implementation is from one mathematical representation to another and so should be able to be entirely automated. Other, more practical work concentrates on the automatic generation of tests from output of earlier activities which can be performed on a piece of code to verify that it behaves correctly.

**Integration and testing**

Once enough components have been implemented and individually tested, they must be integrated as described in the architectural design. Further testing is done to ensure correct behavior and acceptable use of any shared resources. It is also possible at this time to perform some acceptance testing with the customers to ensure that the system meets their requirements. It is only after acceptance of the integrated system that the product is finally released to the customer.

**Maintenance**

After product release, all work on the system is considered under the category of maintenance, until such time as a new version of the product demands a total redesign or the product is phased out entirely. Consequently, the majority of the lifetime of a product is spent in the maintenance activity. Maintenance involves the correction of errors in the systems which are discovered after release and the revision of the system services to satisfy requirements that were not realized during previous development.

**2.6.2 Validation and verification**

Throughout the life cycle, the design must be checked to ensure that it both satisfies the high-level requirements agreed with the customer and is also complete and internally consistent. These checks are referred to as validation and verification, respectively. Verification of a design will most often occur within a single life-cycle activity or between two adjacent activities. For example, in the detailed design of a component of a payroll accounting system, the designer will be concerned with the correctness of the algorithm to compute taxes deducted from an employee‗s gross income.

The architectural design will have provided a general specification of the information input to this component and the information it should output. The detailed description will introduce more information in refining the general specification. The detailed design may also have to change the representations for the information and will almost certainly break up a single high-level operation into several low-level operations that can eventually be implemented. In introducing these changes to information and operations, the designer must show that the refined description is a legal one within its language (internal consistency) and that it describes all of the specified behavior of the high-level description (completeness) in a provably correct way (relative consistency). Validation of a design demonstrates that within the various activities the customer‗s requirements are satisfied.

Validation is a much more subjective exercise than verification, mainly because the disparity between the language of the requirements and the language of the design forbids any objective form of proof. In interactive system design, the validation against HCI requirements is often referred to as evaluation and can be performed by the designer in isolation or in cooperation with the customer.

**2.6.3 Management and contractual issues**

The life cycle described above concentrated on the more technical features of software development. In a technical discussion, managerial issues of design, such as time constraints and economic forces, are not as important. The different activities of the life cycle are logically related to each other. We can see that requirements for a system precede the high- level architectural design which precedes the detailed design, and so on. In reality, it is quite possible that some detailed design is attempted before all of the architectural design. In management, a much wider perspective must be adopted which takes into account the marketability of a system, its training needs, the availability of skilled personnel or possible subcontractors, and other topics outside the activities for the development of the isolated system.

**2.6.4 Interactive systems and the software life cycle**

The life cycle for development we described above presents the process of design in a somewhat pipeline order. In reality, even for batch-processing systems, the actual design process is iterative, work in one design activity affecting work in any other activity either before or after it in the life cycle.

A final point about the traditional software life cycle is that it does not promote the use of notations and techniques that support the user‗s perspective of the interactive system. We discussed earlier the purpose of validation and the formality gap. It is very difficult for an expert on human cognition to predict the cognitive demands that an abstract design would require of the intended user if the notation for the design does not reflect the kind of information the user must recall in order to interact. The same holds for assessing the timing behavior of an abstract design that does not explicitly mention the timing characteristics of the operations to be invoked or their relative ordering. Though no structured development process will entirely eliminate the formality gap, the particular notations used can go a long way towards making validation of non-functional requirements feasible with expert assistance. In the remaining sections of this chapter, we will describe various approaches to augment the design process to suit better the design of interactive systems. These approaches are categorized under the banner of user-centered design.