Chapter 6

The process of interaction design

6.1. Introduction

Design is a practical and creative activity, the ultimate intent of which is to develop a product that helps its users achieve their goals. In previous chapters, we looked at different kinds of interactive products, issues you need to take into account when doing interaction design and some of the theoretical basis for the field. This chapter is the first of four that will explore ***how*** we can design and build interactive products.

Chapter 1 defined interaction design as being concerned with "designing interactive products to support people in their everyday and working lives." But how do you go about doing this?

Developing a product must begin with gaining some understanding of what is required of it, but where do these requirements come from? Whom do you ask about them? Underlying good interaction design is the philosophy of user-centered design, i.e., involving users throughout development, but who are the users? Will they know what they want or need even if we can find them to ask? For an innovative product, users are unlikely to be able to envision what is possible, so where do these ideas come from?

In this chapter, we raise and answer these kinds of questions and discuss the four basic activities and key characteristics of the interaction design process that

were introduced in Chapter 1. We also introduce a lifecycle model of interaction design that captures these activities and characteristics.

The main aims of this chapter are to:

- Consider what 'doing' interaction design involves.
- Ask and provide answers for some important questions about the interaction design process.
- Introduce the idea of a lifecycle model to represent a set of activities and how they are related.
- Describe some lifecycle models from software engineering and HCI and discuss how they relate to the process of interaction design.
- Present a lifecycle model of interaction design.

6.2 What is interaction design about?

There are many fields of design, for example graphic design, architectural design, industrial and software design. Each discipline has its own interpretation of "designing." We are not going to debate these different interpretations here, as we are focussing on interaction design, but a general definition of "design" is informative in beginning to understand what it's about. The definition of design from the Oxford English Dictionary captures the essence of design very well: "(design is) a plan or scheme conceived in the mind and intended for subsequent execution." The act of designing therefore involves the development of such a plan or scheme. For the plan or scheme to have a hope of ultimate execution, it has to be informed with knowledge about its use and the target domain, together with practical constraints such as materials, cost, and feasibility. For example, if we conceived of a plan for building multi-level roads in order to overcome traffic congestion, before the plan could be executed we would **have** to consider drivers' attitudes to using such a construction, the viability of the structure, engineering constraints affecting its feasibility, and cost concerns.

In interaction design, we investigate the artifact's use and target domain by taking a user-centered approach to development. This means that users' concerns direct the development rather than technical concerns.

Design is also about trade-offs, about balancing conflicting requirements. If we take the roads plan again, there may be very strong environmental arguments for stacking roads higher (less countryside would be destroyed), but these must be balanced against engineering and financial limitations that make the proposition less attractive. Getting the balance right requires experience, but it also requires the development and evaluation of alternative solutions. Generating alternatives is a key principle in most design disciplines, and one that should be encouraged in interaction design. As Marc Rettig suggested: "To get a good idea, get lots of ideas" (Rettig, 1994). However, this is not necessarily easy, and unlike many design disciplines, interaction designers are not generally trained to generate alternative designs. However, the ability to brainstorm and contribute alternative ideas can be learned, and techniques from other design disciplines can be successfully used in interaction

design. For example, Danis and Boies (2000) found that using techniques from graphic design that encouraged the generation of alternative designs stimulated innovative interactive systems design. See also the interview with Gillian Crampton Smith at the end of this chapter for her views on how other aspects of traditional design can help produce good interaction design.

Although possible, it is unlikely that just one person will be involved in developing and using a system and therefore the plan must be communicated. This requires it to be captured and expressed in some suitable form that allows review, revision, and improvement. There are many ways of doing this, one of the simplest being to produce a series of sketches. Other common approaches are to write a description in natural language, to draw a series of diagrams, and to build prototypes. A combination of these techniques is likely to be the most effective. When users are involved, capturing and expressing a design in a suitable format is especially important since they are unlikely to understand jargon or specialist notations. In fact, a form that users can interact with is most effective, and building prototypes of one form or another (see Chapter 8) is an extremely powerful approach.

So interaction design involves developing a plan which is informed by the product's intended use, target domain, and relevant practical considerations. Alternative designs need to be generated, captured, and evaluated by users. For the evaluation to be successful, the design must be expressed in a form suitable for users to interact with.

ACTIVITY 6.1 Imagine that you want to design an electronic calendar or diary for yourself. You might use this system to plan your time, record meetings and appointments, mark down people's birthdays, and so on, basically the kinds of things you might do with a paper-based calendar. Draw a sketch of the system outlining its functionality and its general look and feel. Spend about five minutes on this.

Having produced an outline, now spend five minutes reflecting on how you went about tackling this activity. What did you do first? Did you have any particular artifacts or experience to base your design upon? What process did you go through?

Comment The sketch I produced is shown in Figure 6.1. As you can see, I was quite heavily influenced by the paper-based books I currently use! I had in mind that this calendar should allow me to record meetings and appointments, so I need a section representing the days and months. But I also need a section to take notes. I am a prolific note-taker, and so for me this was a key requirement. Then I began to wonder about how I could best use hyperlinks. I certainly want to keep addresses and telephone numbers in my calendar, so maybe there could be a link between, say, someone's name in the calendar and their entry in my address book that will give me their contact details when I need them? But I still want the ability to be able to turn page by page, for when I'm scanning or thinking about how to organize my time. A search facility would be useful too.

The first thing that came into my head when I started doing this was my own paper-based book where I keep appointments, maps, telephone numbers, and other small notes. I also thought about my notebook and how convenient it would be to have the two combined. Then I sat and sketched different ideas about how it might look (although I'm not very good at sketching). The sketch in Figure 6.1 is the version I'm happiest with. Note that my sketch

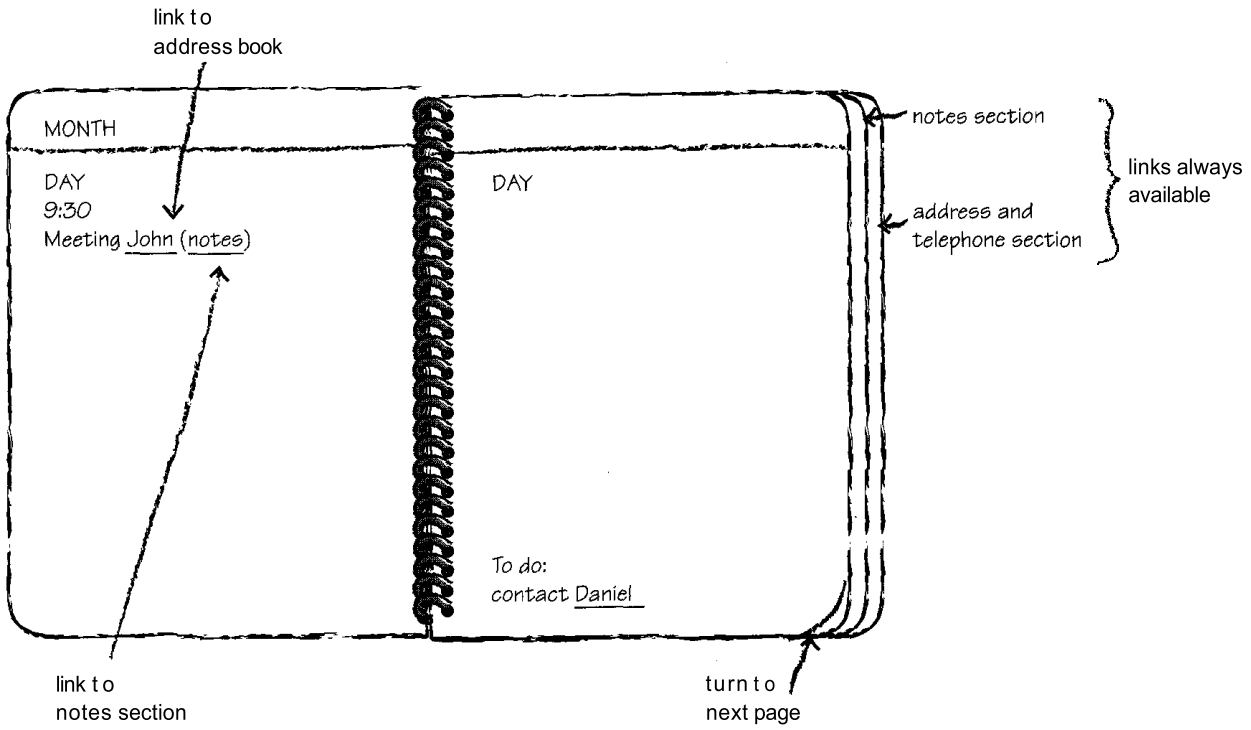


Figure 6.1 An outline sketch of an electronic calendar.

has a strong resemblance to a paper-based book, yet I've also tried to incorporate electronic capabilities. Maybe once I have evaluated this design and ensured that the tasks I want to perform are supported, then I will be more receptive to changing the look away from a paper-based "look and feel."

The exact steps taken to produce a product will vary from designer to designer, from product to product, and from organization to organization. In this activity, you may have started by thinking about what you'd like such a system to do for you, or you may have been **thinking** about **an existing** paper calendar. You may have mixed together features of different systems or other record-keeping support. Having got or arrived at an idea of what you wanted, maybe you then imagined what it might look like, either through sketching with paper and pencil or in your mind.

6.2.1 Four basic activities of interaction design

Four basic activities for interaction design were introduced in Chapter 1, some of which you will have engaged in when doing Activity 6.1. These are: identifying needs and establishing requirements, developing alternative designs that meet those requirements, building interactive versions so that they can be communicated and assessed, and evaluating them, i.e., measuring their acceptability. They are fairly generic activities and can be found in other designs disciplines too. For example, in architectural design (RIBA, 1988) basic requirements are established in a work stage called "inception", alternative design options are considered in a "feasibility" stage and "the brief" is developed through outline proposals and scheme de-

sign. During this time, prototypes may be built or perspectives may be drawn to give clients a better indication of the design being developed. Detail design specifies all components, and working drawings are produced. Finally, the job arrives on site and building commences.

We will be expanding on each of the basic activities of interaction design in the next two chapters. Here we give only a brief introduction to each.

Identifying needs and establishing requirements

In order to design something to support people, we must know who our target users are and what kind of support an interactive product could usefully provide. These needs form the basis of the product's requirements and underpin subsequent design and development. This activity is fundamental to a user-centered approach, and is very important in interaction design; it is discussed further in Chapter 7.

Developing alternative designs

This is the core activity of designing: actually suggesting ideas for meeting the requirements. This activity can be broken up into two sub-activities: conceptual design and physical design. Conceptual design involves producing the conceptual model for the product, and a conceptual model describes what the product should do, behave and look like. Physical design considers the detail of the product including the colors, sounds, and images to use, menu design, and icon design. Alternatives are considered at every point. You met some of the ideas for conceptual design in Chapter 2; we go into more detail about conceptual and physical design in Chapter 8.

Building interactive versions of the designs

Interaction design involves designing interactive products. The most sensible way for users to evaluate such designs, then, is to interact with them. This requires an interactive version of the designs to be built, but that does not mean that a software version is required. There are different techniques for achieving "interaction," not all of which require a working piece of software. For example, paper-based prototypes are very quick and cheap to build and are very effective for identifying problems in the early stages of design, and through role-playing users can get a real sense of what it will be like to interact with the product. This aspect is also covered in Chapter 8.

Evaluating designs

Evaluation is the process of determining the usability and acceptability of the product or design that is measured in terms of a variety of criteria including the number of errors users make using it, how appealing it is, how well it matches the requirements, and so on. Interaction design requires a high level of user involvement throughout development, and this enhances the chances of an acceptable product being delivered. In most design situations you will find a number of activities concerned with

quality assurance and testing to make sure that the final product is "fit-for-purpose." Evaluation does not replace these activities, but complements and enhances them. We devote Chapters 10 through 14 to the important subject of evaluation.

The activities of developing alternative designs, building interactive versions of the design, and evaluation are intertwined: alternatives are evaluated through the interactive versions of the designs and the results are fed back into further design. This iteration is one of the key characteristics of the interaction design process, which we introduced in Chapter 1.

6.2.2 Three key characteristics of the interaction design process

There are three characteristics that we believe should form a key part of the interaction design process. These are: a user focus, specific usability criteria, and iteration.

The need to ***focus on users*** has been emphasized throughout this book, so you will not be surprised to see that it forms a central plank of our view on the interaction design process. While a process cannot, in itself, guarantee that a development will involve users, it can encourage focus on such issues and provide opportunities for evaluation and user feedback.

Specific usability and user experience goals should be identified, clearly documented, and agreed upon at the beginning of the project. They help designers to choose between different alternative designs and to check on progress as the product is developed.

Iteration allows designs to be refined based on feedback. As users and designers engage with the domain and start to discuss requirements, needs, hopes and aspirations, then different insights into what is needed, what will help, and what is feasible will emerge. This leads to a need for iteration, for the activities to inform each other and to be repeated. However good the designers are and however clear the users may think their vision is of the required artifact, it will be necessary to revise ideas in light of feedback, several times. This is particularly true if you are trying to innovate. Innovation rarely emerges whole and ready to go. It takes time, evolution, trial and error, and a great deal of patience. Iteration is inevitable because designers never get the solution right the first time (Gould and Lewis, 1985).

We shall return to these issues and expand upon them in Chapter 9.

6.3 Some practical issues

Before we consider hbw the activities and key characteristics of interaction design can be pulled together into a coherent process, we want to consider some questions highlighted by the discussion so far. These questions must be answered if we are going to be able to "do" interaction design in practice. These are:

- Who are the users?
- What do we mean by needs?
- How do you generate alternative designs?
- How do you choose among alternatives?

6.3.1 Who are the users?

In Chapter 1, we said that an overarching objective of interaction design is to optimize the interactions people have with computer-based products, and that this requires us to support needs, match wants, and extend capabilities. We also stated above that the activity of identifying these needs and establishing requirements was fundamental to interaction design. However, we can't hope to get very far with this intent until we know who the users are and what they want to achieve. As a starting point, therefore, we need to know who we consult to find out the users' requirements and needs.

Identifying the users may seem like a straightforward activity, but in fact there are many interpretations of "user." The most obvious definition is those people who interact directly with the product to achieve a task. Most people would agree with this definition; however, there are others who can also be thought of as users. For example, Holtzblatt and Jones (1993) include in their definition of "users" those who manage direct users, those who receive products from the system, those who test the system, those who make the purchasing decision, and those who use competitive products. Eason (1987) identifies three categories of user: primary, secondary and tertiary. Primary users are those likely to be frequent hands-on users of the system; secondary users are occasional users or those who use the system through an intermediary; and tertiary users are those affected by the introduction of the system or who will influence its purchase.

The trouble is that there is a surprisingly wide collection of people who all have a stake in the development of a successful product. These people are called ***stakeholders.*** Stakeholders are "people or organizations who will be affected by the system and who have a direct or indirect influence on the system requirements" (Kotonya and Sommerville, 1998). Dix et al. (1993) make an observation that is very pertinent to a user-centered view of development, that "It will frequently be the case that the formal 'client' who orders the system falls very low on the list of those affected. Be very wary of changes which take power, influence or control from some stakeholders without returning something tangible in its place."

Generally speaking, the group of stakeholders for a particular product is going to be larger than the group of people you'd normally think of as users, although it will of course include users. Based on the definition above, we can see that the group of stakeholders includes the development team itself as well as its managers, the direct users and their managers, recipients of the product's output, people who may lose their jobs because of the introduction of the new product, and so on.

For example, consider again the calendar system in Activity 6.1. According to the description we gave you, the user group for the system has just one member: you. However, the stakeholders for the system would also include people you make appointments with, people whose birthdays you remember, and even companies that produce paper-based calendars, since the introduction of an electronic calendar may increase competition and force them to operate differently.

This last point may seem a little exaggerated for just one system, but if you think of others also migrating to an electronic version, and abandoning their paper calendars, then you can see how the companies may be affected by the introduction of the system.

The net of stakeholders is really quite wide! We do not suggest that you need to involve all of the stakeholders in your user-centered approach, but it is important to be aware of the wider impact of any product you are developing. Identifying the stakeholders for your project means that you can make an informed decision about who should be involved and to what degree.

ACTIVITY 6.2 Who do you think are the stakeholders for the check-out system of a large supermarket?

Comment First, there are the check-out operators. These are the people who sit in front of the machine and pass the customers' purchases over the bar code reader, receive payment, hand over receipts, etc. Their stake in the success and usability of the system is fairly clear and direct. Then you have the customers, who want the system to work properly so that they are charged the right amount for the goods, receive the correct receipt, are served quickly and efficiently. Also, the customers want the check-out operators to be satisfied and happy in their work so that they don't have to deal with a grumpy assistant. Outside of this group, you then have supermarket managers and supermarket owners, who also want the assistants to be happy and efficient and the customers to be satisfied and not complaining. They also don't want to lose money because the system can't handle the payments correctly. Other people who will be affected by the success of the system include other supermarket employees such as warehouse staff, supermarket suppliers, supermarket owners' families, and local shop owners whose business would be affected by the success or failure of the system. We wouldn't suggest that you should ask the local shop owner about requirements for the supermarket check-out system. However, you might want to talk to warehouse staff, especially if the system links in with stock control or other functions.

6.3.2 What do we mean by "needs"?

If you had asked someone in the street in the late 1990s what she 'needed', I doubt that the answer would have included interactive television, or a jacket which was wired for communication, or a smart fridge. If you presented the same person with these possibilities and asked whether she would buy them if they were available, then the answer would have been different. When we talk about identifying needs, therefore, it's not simply a question of asking people, "What do you need?" and then supplying it, because people don't necessarily know what is possible (see Suzanne Robertson's interview at the end of Chapter 7 for "un-dreamed-of" requirements). Instead, we have to approach it by understanding the characteristics and capabilities of the users, what they are trying to achieve, how they achieve it currently, and whether they would achieve their goals more effectively if they were supported differently.

There are many dimensions along which a user's capabilities and characteristics may vary, and that will have an impact on the product's design. You have met

some of these in Chapter 3. For example, a person's physical characteristics may affect the design: size of hands may affect the size and positioning of input buttons, and motor abilities may affect the suitability of certain input and output devices; height is relevant in designing a physical kiosk, for example; and strength in designing a child's toy—a toy should not require too much strength to operate, but may require strength greater than expected for the target age group to change batteries or perform other operations suitable only for an adult. Cultural diversity and experience may affect the terminology the intended user group is used to, or how nervous about technology a set of users may be.

If a product is a new invention, then it can be difficult to identify the users and representative tasks for them; e.g., before microwave ovens were invented, there were no users to consult about requirements and there were no representative tasks to identify. Those developing the oven had to imagine who might want to use such an oven and what they might want to do with it.

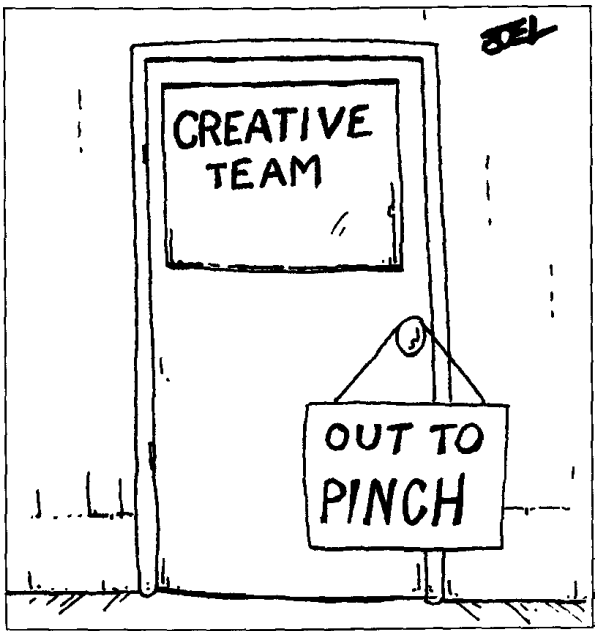
It may be tempting for designers simply to design what they would like, but their ideas would not necessarily coincide with those of the target user group. It is *imperative* that representative users from the real target group be consulted. For example, a company called Netpliance was developing a new "Internet appliance," i.e., a product that would seamlessly integrate all the services necessary for the user to achieve a specific task on the Internet (Isensee et al., 2000). They took a user-centered approach and employed focus group studies and surveys to understand their customers' needs. The marketing department led these efforts, but developers observed the focus groups to learn more about their intended user group. Isensee et al. (p. 60) observe that "It is always tempting for developers to create products they would want to use or similar to what they have done before. However, in the Internet appliance space, it was essential to develop for a new audience that desires a simpler product than the computer industry has previously provided."

In these circumstances, a good indication of future behavior is current or past behavior. So it is always useful to start by understanding similar behavior that is already established. Apart from anything else, introducing something new into people's lives, especially a new "everyday" item such as a microwave oven, requires a culture change in the target user population, and it takes a long time to effect a culture change. For example, before cell phones were so widely available there were no users and no representative tasks available for study, *per* se. But there were standard telephones and so understanding the tasks people perform with, and in connection with, standard telephones was a useful place to start. Apart from making a telephone call, users also look up people's numbers, take messages for others not currently available, and find out the number of the last person to ring them. These kinds of behavior have been translated into memories for the telephone, answering machines, and messaging services for mobiles. In order to maximize the benefit of e-commerce sites, traders have found that referring back to customers' non-electronic habits and behaviors can be a good basis for enhancing e-commerce activity (CHI panel, 2000; Lee et al., 2000).

6.3.3 How do you generate alternative designs?

A common human tendency is to stick with something that we know works. We probably recognize that a better solution may exist out there somewhere, but it's very easy to accept this one because we know it works—it's "good enough." Settling for a solution that is good enough is not, in itself, necessarily "bad," but it may be undesirable because good alternatives may never be considered, and considering alternative solutions is a crucial step in the process of design. But where do these alternative ideas come from?

One answer to this question is that they come from the individual designer's flair and creativity. While it is certainly true that some people are able to produce wonderfully inspired designs while others struggle to come up with any ideas at all, very little in this world is completely new. Normally, innovations arise through cross-fertilization of ideas from different applications, the evolution of an existing product through use and observation, or straightforward copying of other, similar products. For example, if you think of something commonly believed to be an "invention," such as the steam engine, this was in fact inspired by the observation that the steam from a kettle boiling on the stove lifted the lid. Clearly there was an amount of creativity and engineering involved in making the jump from a boiling kettle to a steam engine, but the kettle provided the inspiration to translate experience gained in one context into a set of principles that could be applied in another. As an example of evolution, consider the word processor. The capabilities of suites of office software have gradually increased from the time they first appeared. Initially, a word processor was just an electronic version of a typewriter, but gradually other capabilities, including the spell-checker, thesaurus, style sheets, graphical capabilities, etc., were added.



So although creativity and invention are often wrapped in mystique, we do understand something of the process and of how creativity can be enhanced or inspired. We know, for instance, that browsing a collection of designs will inspire designers to consider alternative perspectives, and hence alternative solutions. The field of case-based reasoning (Maher and Pu, 1997) emerged from the observation that designers solve new problems by drawing on knowledge gained from solving previous similar problems. As Schank (1982; p. 22) puts it, "An expert is someone who gets reminded of just the right prior experience to help him in processing his current experiences." And while those experiences may be the designer's own, they can equally well be others'.

A more pragmatic answer to this question, then, is that alternatives come from looking at other, similar designs, and the process of inspiration and creativity can be enhanced by prompting a designer's own experience and by looking at others' ideas and solutions. Deliberately seeking out suitable sources of inspiration is a valuable step in any design process. These sources may be very close to the intended new product, such as competitors' products, or they may be earlier versions of similar systems, or something completely different.

ACTIVITY 6.3 Consider again the calendar system introduced at the beginning of the chapter. Reflecting on the process again, what do you think inspired your outline design? See if you can identify any elements within it that you believe are truly innovative.

Comment For my design, I haven't seen an electronic calendar, although I have seen plenty of other software-based systems. My main sources of inspiration were my current paper-based books.

Some of the things you might have been thinking of include your existing paper-based calendar, and other pieces of software you commonly use and find helpful or easy to use in some way. Maybe you already have access to an electronic calendar, which will have given you some ideas, too. However, there are probably other aspects that make the design somehow unique to you and may be innovative to a greater or lesser degree.

All this having been said, under some circumstances the scope to consider alternative designs may be limited. Design is a process of balancing constraints and constantly trading off one set of requirements with another, and the constraints may be such that there are very few viable alternatives available. As another example, if you are designing a software system to run under the Windows operating system, then elements of the design will be prescribed because you must conform to the Windows "look and feel," and to other constraints intended to make Windows programs consistent for the user. We shall return to style guides and standards in Chapter 8.

If you are producing an upgrade to an existing system, then you may face other constraints, such as wanting to keep the familiar elements of it and retain the same "look and feel." However, this is not necessarily a rigid rule. Kent Sullivan reports that when designing the Windows 95 operating system to replace the Windows 3.1 and Windows for Workgroups 3.11 operating systems, they initially focused too much on consistency with the earlier versions (Sullivan, 1996).

BOX 6.1 A Box Full of Ideas

The innovative product design company IDEO was introduced in Chapter 1. It has been involved in the development of many artifacts including the first commercial computer mouse and the PalmPilot V. Underlying some of their creative flair is a collection of weird and wonderful engineering housed in a large flatbed filing cabinet called the TechBox (see Figure 6.2). The TechBox holds around 200 gizmos and interesting materials, divided into categories: "Amazing Materials," "Cool Mechanisms," "Interesting Manufacturing Processes," "Electronic Technologies," and "Thermal and Optical." Each item has been placed in the box because it represents a neat idea or a new process. Staff at IDEO take along a selection of items from the TechBox to brainstorming meetings. The items may be chosen because they provide useful visual props or possible solutions to a particular issue, or simply to provide some light relief.

Each item is clearly labeled with its name and category, but further information can be found by accessing the TechBox's online catalog. Each item has its own page detailing what the item is, why it's interesting, where it came from, and who has used it or knows more about it. For example, the page in Figure 6.3 relates to a metal injection-molding technique.

Other items in the box include an example of metal-coated wood, materials with and without holes that stretch, bend, and change shape or color at different temperatures.

Each TechBox has its own curator who is responsible for maintaining and cataloging the items and for promoting its use within the office. Anyone can submit a new item for consideration and

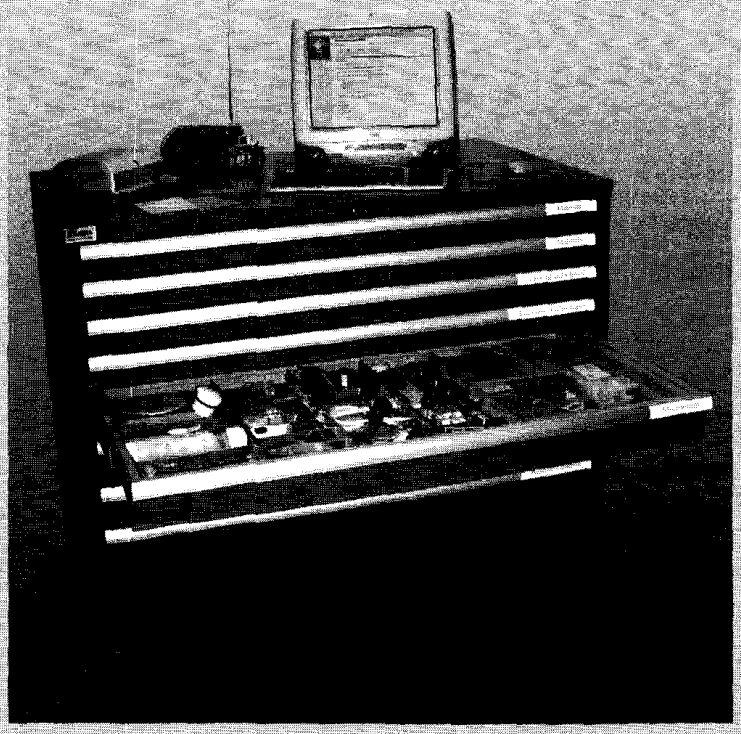


Figure 6.2 The TechBox at IDEO.

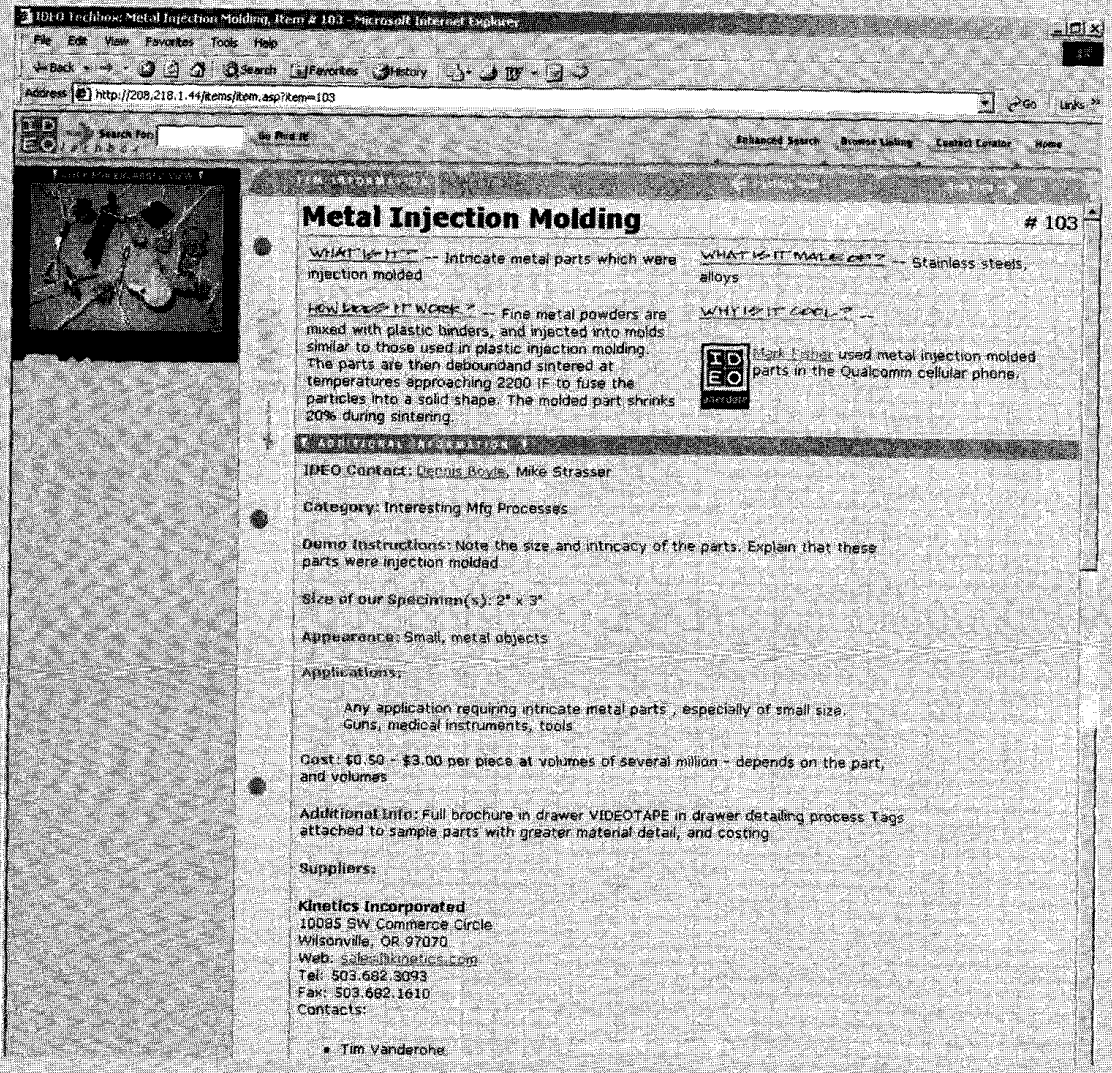


Figure 6.3 The web page for the metal injection molding.

as items become common place, they are removed from the TechBox to make way for the next generation of fascinating contraptions.

How are these things used? Well here is one example from Patrick Hall at the London IDEO office (see Figure 6.4):

> IDEO was asked to review the design of a mass-produced hand-held medical product that was deemed to be too big.

As well as brainstorming and other conventional idea-generation methods, I was able to immediately pick out items which I knew about from having used the TechBox in the past: Deep Draw; Fibre-Optic magnifier; Metal Injection molding; Flexy Battery. Further browsing and searching using the keywords search engine highlighted in-mold assembly and light-intensifying film. The

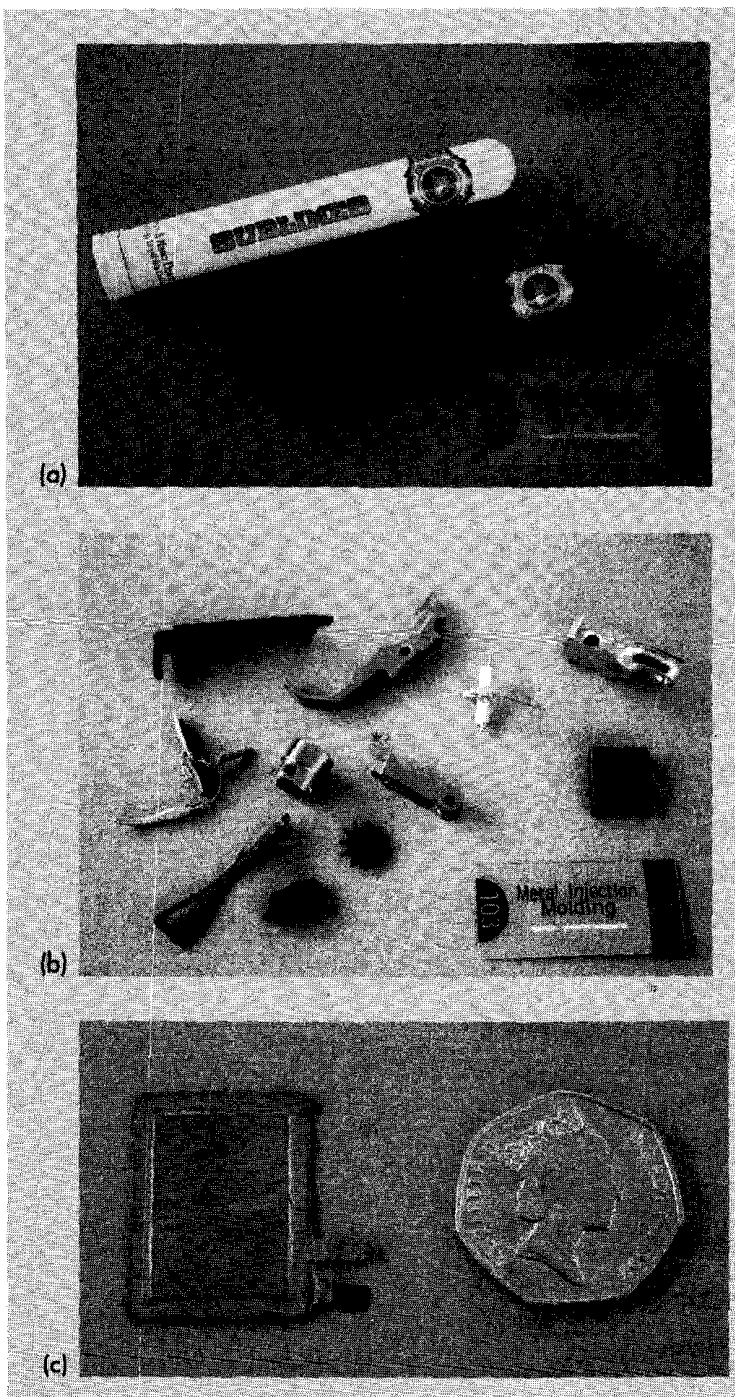


Figure 6.4 Items from the TechBox used in the design of a medical product. (a) Deep Draw—A metal-forming process to generate close-ended cylindrical parts; (b) Metal Injection Molding—A molding and sintering process to produce complex metal parts in high numbers; (c) Flexy Battery—a lithium polymer cell from Varta that is very thin (intended for Smart Cards) and can be formed into cylindrical shapes.

associated web pages for these items enabled me to learn more about these items immediately and indicated who to talk to in IDEO to find out more, and the details of vendors to approach.

The project ended at the feasibility phase, with the client pursuing the technologies I had suggested. Only the fiber-optic magnifier proved (immediately) not to be worth pursuing (because of cost).

DILEMMA Copying for Inspiration: Is It Legal?

Designers draw on their experience of design when approaching a new project. This includes the use of previous designs that they know work, both designs they have created themselves and those that others have created. Others' creations often spark inspiration that also leads to new ideas and innovation. This is well known and understood. However, the expression of an idea is protected by copyright, and people who infringe that copyright can be taken to court and prosecuted. Note that copyright covers the expression of an idea and not the idea itself. This means, for example, that while there are numerous word processors all with similar functionality, this does not represent an infringement of copyright as the idea has been expressed in different ways, and it's the expression that's been copyrighted. Copyright is free and is automatically invested in the author of something, e.g., the writer of a book or a programmer who develops a program, unless he signs the copyright over to someone else. Authors writing for academic journals often are asked to sign over their copyright to the publisher of the journal. Various limitations and special conditions can apply, but basically, the copyright is no longer theirs. People who produce something through their employment, such as programs or products, may have in their employment contract a statement saying that the copyright relating to anything produced in the course of that employment is automatically assigned to the employer and does not remain with the employee.

On the other hand, patenting is an alternative to copyright that does protect the idea rather than the expression. There are various forms of patenting, each of which is designed to allow the inventor the chance to capitalize on an idea. It is unusual for software to be patented, since it is a long, slow, and expensive process, although there is a recent trend towards patenting business processes. For example, Amazon, the on-line bookstore, has patented its "one-click" purchasing process, which allows regular users simply to choose a book and buy it with one mouse click (US Patent No. 5960411, September 29, 1999). This is possible because the system stores its customers' details and "recognizes" them when they access the site again.

So the dilemma comes in knowing when it's OK to use someone else's work as a source of inspiration and when you are infringing copyright or patent law. The issues around this question are complex and detailed, and well beyond the scope of this book, but more information and examples of law cases that have been brought successfully and unsuccessfully can be found in Bainbridge (1999).

6.3.4 How do you choose among alternative designs?

Choosing among alternatives is about making design decisions: Will the device use keyboard entry or a touch screen? Will the device provide an automatic memory function or not? These decisions will be informed by the information gathered about users and their tasks, and by the technical feasibility of an idea. Broadly speaking, though, the decisions fall into two categories: those that are about externally visible and measurable features, and those that are about characteristics internal to the system that cannot be observed or measured without dissecting it. For example, externally visible and measurable factors for a building design include the ease of access to the building, the amount of natural light in rooms, the width of corridors, and the number of power outlets. In a photocopier, externally visible and measurable factors include the physical size of the machine, the speed and quality of copying, the different sizes of paper it can use, and so on. Underlying each of these factors are other considerations that cannot be observed or studied without dissecting the building or the machine. For example, the number of

power outlets will be dependent on how the wiring within the building is designed and the capacity of the main power supply; the choice of materials used in a photocopier may depend on its friction rating and how much it deforms under certain conditions.

In an interactive product there are similar factors that are externally visible and measurable and those that are hidden from the users' view. For example, exactly why the response time for a query to a database (or a web page) is, say, 4 seconds will almost certainly depend on technical decisions made when the database was constructed, but from the users' viewpoint the important observation is the fact that it does take 4 seconds to respond.

In interaction design, the way in which the users interact with the product is considered the driving force behind the design and so we concentrate on the externally visible and measurable behavior. Detailed internal workings are important only to the extent that they affect the external behavior. This does not mean that design decisions concerning a system's internal behavior are any less important: however, the tasks that the user will perform should influence design decisions no less than technical issues.

So, one answer to the question posed above is that we choose between alternative designs by letting users and stakeholders interact with them **and** by **discussing** their experiences, preferences and suggestions for improvement. This is fundamental to a user-centered approach to development. This in turn means that the designs must be available in a form that can be reasonably evaluated with users, not in technical jargon or notation that seems impenetrable to them.

One form traditionally used for communicating a design is documentation, e.g., a description of how something will work or a diagram showing its components. The trouble is that a static description cannot capture the dynamics of behavior, and for an interaction device we need to communicate to the users what it will be like to actually operate it.

In many design disciplines, prototyping is used to overcome potential client misunderstandings and to test the technical feasibility of a suggested design and its production. Prototyping involves producing a limited version of the product with the purpose of answering specific questions about the design's feasibility or appropriateness. Prototypes give a better impression of the user experience than simple descriptions can ever do, and there are different kinds of prototyping that are suitable for different stages of development and for eliciting different kinds of information. One experience illustrating the benefits of prototyping is described in Box 6.2. So one important aspect of choosing among alternatives is that prototypes should be built and evaluated by users. We'll revisit the issue of prototyping in Chapter 8.

Another basis on which to choose between alternatives is "quality," but this requires a clear understanding of what "quality" means. People's views of what is a quality product vary, and we don't always write it down. Whenever we use anything we have some notion of the level of quality we are expecting, wanting, or needing. Whether this level of quality is expressed formally or informally does not matter. The point is that it exists and we use it consciously or subconsciously to evaluate alternative items. For example, if you have to wait too long to download

BOX 6.2 The Value of Prototyping

I learned the value of a prototype through a very effective role-playing exercise. I was on a course designed to introduce new graduates to different possible careers in industry. One of the themes was production and manufacturing and the aim of one group exercise was to produce a notebook. Each group was told that it had 30 minutes to deliver 10 books to the person in charge. Groups were given various pieces of paper, scissors, sticky tape, staples, etc., and told to organize ourselves as best we could. So my group set to work organizing ourselves into a production line, with one of us cutting up the paper, another stapling the pages together, another sealing the binding with the sticky tape, and so on. One person was even in charge of quality assurance. It took us less than 10 minutes to produce the 10 books, and we rushed off with our delivery. When we showed the person in charge, he replied, "That's not what I wanted, I need it bigger than that." Of course, the size of the notebook wasn't specified in the description of the task, so we found out how big he wanted it, got some more materials, and scooted back to produce 10 more books. Again, we set up our production line and produced 10 books to the correct size. On delivery we were again told that it was not what was required: he wanted the binding to work the other way around. This time we got as many of the requirements as we could and went back, developed one book, and took that back for further feedback and refinement before producing the 10 required.

If we had used prototyping as a way of exploring our ideas and checking requirements in the first place, we could have saved so much effort and resource!

a web page, then you are likely to give up and try a different site—you are applying a certain measure of quality associated with the time taken to download the web page. If one cell phone makes it easy to perform a critical function while another involves several complicated key sequences, then you are likely to buy the former rather than the latter. You are applying a quality criterion concerned with efficiency.

Now, if you are the only user of a product, then you don't necessarily have to express your definition of "quality" since you don't have to communicate it to anyone else. However, as we have seen, most projects involve many different stakeholder groups, and you will find that each of them has a different definition of quality and different acceptable limits for it. For example, although all stakeholders may agree on targets such as "response time will be fast" or "the menu structure will be easy to use," exactly what each of them means by this is likely to vary. Disputes are inevitable when, later in development, it transpires that "fast" to one set of stakeholders meant "under a second," while to another it meant "between 2 and 3 seconds." Capturing these different views in clear unambiguous language early in development takes you halfway to producing a product that will be regarded as "good" by all your stakeholders. It helps to clarify expectations, provides a benchmark against which products of the development process can be measured, and gives you a basis on which to choose among alternatives.

The process of writing down formal, verifiable–and hence measurable–usability criteria is a key characteristic of an approach to interaction design called ***usability engineering*** that has emerged over many years and with various proponents (Whiteside

et al., 1988; Nielsen, 1993). Usability engineering involves specifying quantifiable measures of product performance, documenting them in a usability specification, and assessing the product against them. One way in which this approach is used is to make changes to subsequent versions of a system based on feedback from carefully documented results of usability tests for the earlier version. We shall return to this idea later when we discuss evaluation.

ACTIVITY 6.4 Consider the calendar system that you designed in Activity 6.1. Suggest some usability criteria that you could use to determine the calendar's quality. You will find it helpful to think in terms of the usability goals introduced in Chapter 1: effectiveness, efficiency, safety, utility, learnability, and memorability. Be as specific as possible. Check your criteria by considering exactly what you would measure and how you would measure its performance.

Having done that, try to do the same thing for the user experience goals introduced in Chapter 1; these relate to whether a system is satisfying, enjoyable, motivating, rewarding, and so on.

Comment

Finding measurable characteristics for some of these is not easy. Here are some suggestions, but you may have found others. Note that the criteria must be measurable and very specific.

- ***Effectiveness:*** Identifying measurable criteria for this goal is particularly difficult since it is a combination of the other goals. For example, does the system support you in keeping appointments, taking notes, and so on. In other words, is the calendar used?
- *Efficiency*: Assuming that there is a search facility in the calendar, what is the response time for finding a specific day or a specific appointment?
- ***Safety:*** How often does data get lost or does the user press the wrong button? This may be measured, for example, as the number of times this happens per hour of use.
- ***Utility:*** How many functions offered by the calendar are used every day, how many every week, how many every month? How many tasks are difficult to complete in a reasonable time because functionality is missing or the calendar doesn't support the right subtasks?
- ***Learnability:*** How long does it take for a novice user to be able to do a series of set tasks, e.g., make an entry into the calendar for the current date, delete an entry from the current date, edit an entry in the following day?
- ***Memorability:*** If the calendar isn't used for a week, how many functions can you remember how to perform? How long does it take you to remember how to perform your most frequent task?

Finding measurable characteristics for the user experience criteria is even harder, though. How do you measure satisfaction, fun, motivation or aesthetics? What is entertaining to one person may be boring to another; these kinds of criteria are subjective, and so cannot be measured objectively.

6.4 Lifecycle models: showing how the activities are related

Understanding what activities are involved in interaction design is the first step to being able to do it, but it is also important to consider how the activities are related

to one another so that the full development process can be seen. The term *lifecycle model*[1] is used to represent a model that captures a set of activities and how they are related. Sophisticated models also incorporate a description of when and how to move from one activity to the next and a description of the deliverables for each activity. The reason such models are popular is that they allow developers, and particularly managers, to get an overall view of the development effort so that progress can be tracked, deliverables specified, resources allocated, targets set, and so on.

Existing models have varying levels of sophistication and complexity. For projects involving only a few experienced developers, a simple process would probably be adequate. However, for larger systems involving tens or hundreds of developers with hundreds or thousands of users, a simple process just isn't enough to provide the management structure and discipline necessary to engineer a usable product. So something is needed that will provide more formality and more discipline. Note that this does not mean that innovation is lost or that creativity is stifled. It just means that a structured process is used to provide a more stable framework for creativity.

However simple or complex it appears, any lifecycle model is a simplified version of reality. It is intended as an abstraction and, as with any good abstraction, only the amount of detail required for the task at hand should be included. Any organization wishing to put a lifecycle model into practice will need to add detail specific to its particular circumstances and culture. For example, Microsoft wanted to maintain a small-team culture while also making possible the development of very large pieces of software. To this end, they have evolved a process that has been called "synch and stabilize," as described in Box 6.3.

In the next subsection, we introduce our view of what a lifecycle model for interaction design might look like that incorporates the four activities and the three key characteristics of the interaction design process discussed above. This will form the basis of our discussion in Chapters 7 and 8. Depending on the kind of system being developed, it may not be possible or appropriate to follow this model for every element of the system, and it is certainly true that more detail would be required to put the lifecycle into practice in a real project.

Many other lifecycle models have been developed in fields related to interaction design, such as software engineering and HCI, and our model is evolved from these ideas. To put our interaction design model into context we include here a description of five lifecycle models, three from software engineering and two from HCI, and consider how they relate to it.

---

[1] Sommerville (2001) uses the term process model to mean what we call a lifecycle model, and refers to the waterfall model as the software lifecycle. Pressman (1992) talks about paradigms. In HCI the term "lifecycle model" is used more widely. For this reason, and because others use "process model" to represent something that is more detailed than a lifecycle model (e.g., Comer, 1997) we have chosen to use lifecycle model.

BOX 6.3 How Microsoft Builds Software *(Cusumano and Selby, 1997)*

Microsoft is one of the largest software companies in the world and builds some very complex software; for example, Windows 95 contains more than 11 million lines of code and required more than 200 programmers. Over a two-and-a-half-year period from the beginning of 1993, two researchers, Michael Cusumano and Richard Selby, were given access to Microsoft project documents and key personnel for study and interview. Their aim was to build up an understanding of how Microsoft produces software. Rather than adopt the structured software engineering practices others have followed, Microsoft's strategy has been to cultivate entrepreneurial flexibility throughout its software teams. In essence, it has tried to scale up the culture of a loosely-structured, small software team. "The objective is to get many small teams (three to eight developers each) or individual programmers to work together as a single relatively large team in order to build large products relatively quickly while still allowing individual programmers and teams freedom to evolve their designs and operate nearly autonomously" (p. 54).

In order to maintain consistency and to ensure that products are eventually shipped, the teams *synchronize* their activities daily and periodically stabilize the whole product. Cusumano and Selby have therefore labeled Microsoft's unique process "synch and stabilize." Figure 6.5 shows an overview of this process, which is divided into three phases: the planning phase, the development phase and the stabilization phase.

The planning phase begins with a vision statement that defines the goals of the new product and the user activities to be supported by the product. (Microsoft uses a method called activity-based planning to identify and prioritize the features to be built; we return to this in Chapter 9.) The program managers together with the developers then write a functional specification in enough detail to describe features and to develop schedules and al-

Planning Phase Define product vision, specifications, and schedule

- **Vision Statement** Product and program management use extensive customer input to identify and priority-order product features.
- **Specification Document** Based on vision statement, program management and development group define feature functionality, architectural issues, and component interdependencies.
- **Schedule and Feature Team Formation** Based on specification document, program management coordinates schedule and arranges feature teams that each contain approximately 1 program manager, 3–8 developers, and 3–8 testers (who work in parallel 1:1 with developers).

Development Phase Feature development in 3 or 4 sequential subprojects that each results in a milestone release

Program managers coordinate evolution of specification. Developers design, code, and debug. Testers pair with developers for continuous testing.

- **Subproject I** First 1/3 of features (Most critical features and shared components)
- **Subproject II** Second 1/3 of features
- **Subproject III** Final 1/3 of features (Least critical features)

Stabilization Phase Comprehensive internal and external testing, final product stabilization, and ship

Program managers coordinate OEMs and ISVs and monitor customer feedback. Developers perform final debugging and code stabilization. Testers recreate and isolate errors.

- **Internal Testing** Thorough testing of complete product within the company.
- **External Testing** Thorough testing of complete product outside the company by "beta" sites, such as OEMs, ISVs, and end users.
- **Release preparation** Prepare final release of "golden master" disks and documentation for manufacturing.

Figure 6.5 Overview of the synch and stabilize development approach.

locate staff. The feature list in this document will change by about 30% during the course of development, so the list is not fixed at this time. In the next phase, the development phase, the feature list is divided into three or four parts, each with its own small development team, and the schedule is divided into sequential subprojects, each with its own deadline (milestone). The teams work in parallel on a set of features and synchronize their work by putting together their code and finding errors on a daily and weekly basis. This is necessary because many programmers may be working on the same code at once. For example, during the peak development of Excel 3.0, 34 developers were actively changing the same source code on a daily basis.

At the end of a subproject, i.e., on reaching a milestone, all errors are found and fixed, thus stabilizing the product, before moving on to the next subproject and eventually to the final milestone, which represents the release date. Figure 6.6 shows an overview of the milestone structure for a project with three subprojects. This synch-and-stabilize approach has been used to develop Excel, Office, Publisher, Windows 95, Windows NT, Word, and Works, among others.

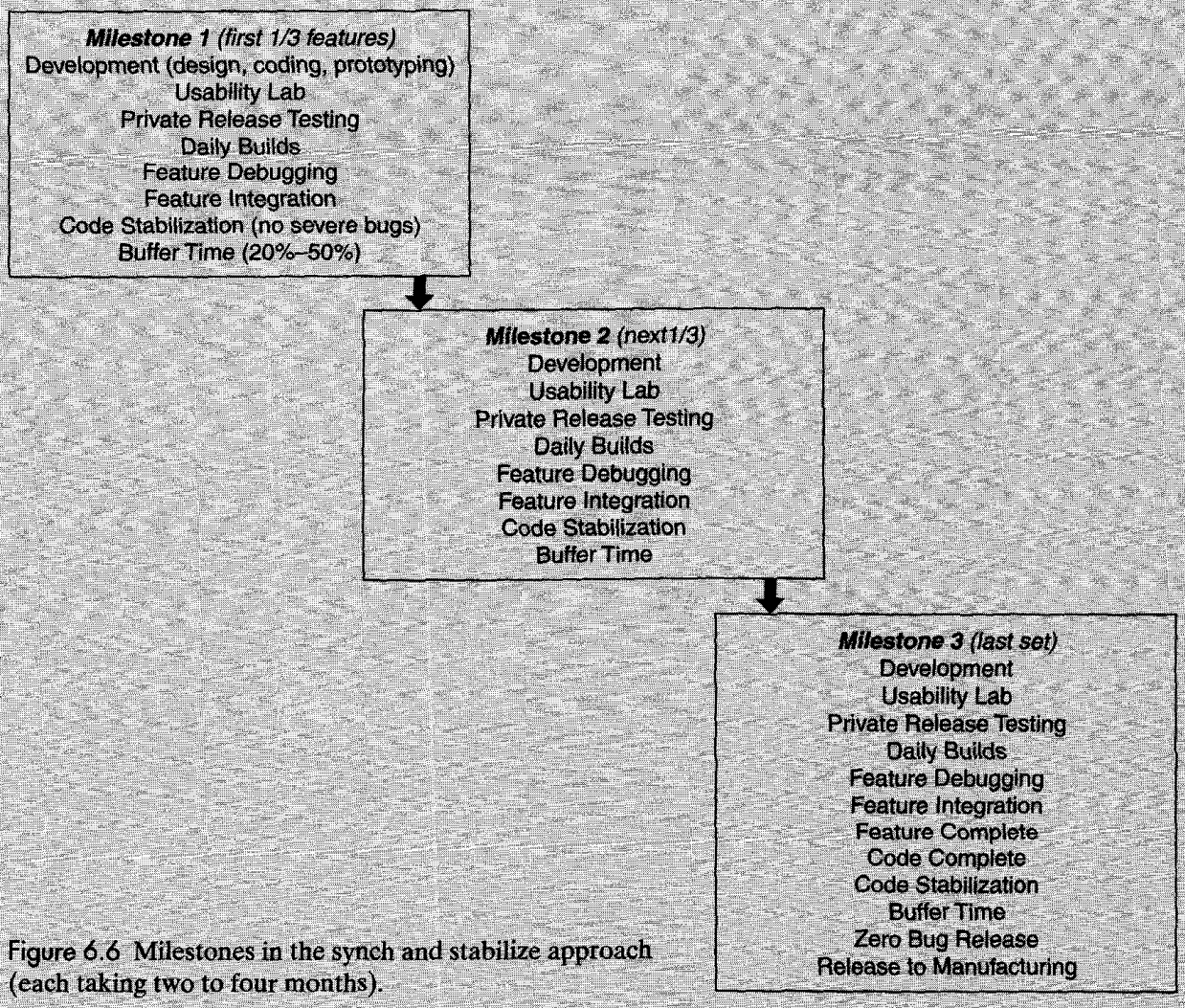


Figure 6.6 Milestones in the synch and stabilize approach (each taking two to four months).

6.4.1 A simple lifecycle model for interaction design

We see the activities of interaction design as being related as shown in Figure 6.7. This model incorporates iteration and encourages a user focus. While the outputs from each activity are not specified in the model, you will see in Chapter 7 that our description of establishing requirements includes the need to identify specific usability criteria.

The model is not intended to be prescriptive; that is, we are not suggesting that this is how all interactive products are or should be developed. It is based on our observations of interaction design and on information we have gleaned in the research for this book. It has its roots in the software engineering and HCI lifecycle models described below, and it represents what we believe is practiced in the field.

Most projects start with identifying needs and requirements. The project may have arisen because of some evaluation that has been done, but the lifecycle of the new (or modified) product can be thought of as starting at this point. From this activity, some alternative designs are generated in an attempt to meet the needs and requirements that have been identified. Then interactive versions of the designs are developed and evaluated. Based on the feedback from the evaluations, the team may need to return to identifying needs or refining requirements, or it may go straight into redesigning. It may be that more than one alternative design follows this iterative cycle in parallel with others, or it may be that one alternative at a time is considered. Implicit in this cycle is that the final product will emerge in an evolutionary fashion from a rough initial idea through to the finished product. Exactly how this evolution happens may vary from project to project, and we return to this issue in Chapter 8. The only factor limiting the number of times through the cycle is the resources available, but whatever the number is, development ends with an evaluation activity that ensures the final product meets the prescribed usability criteria.

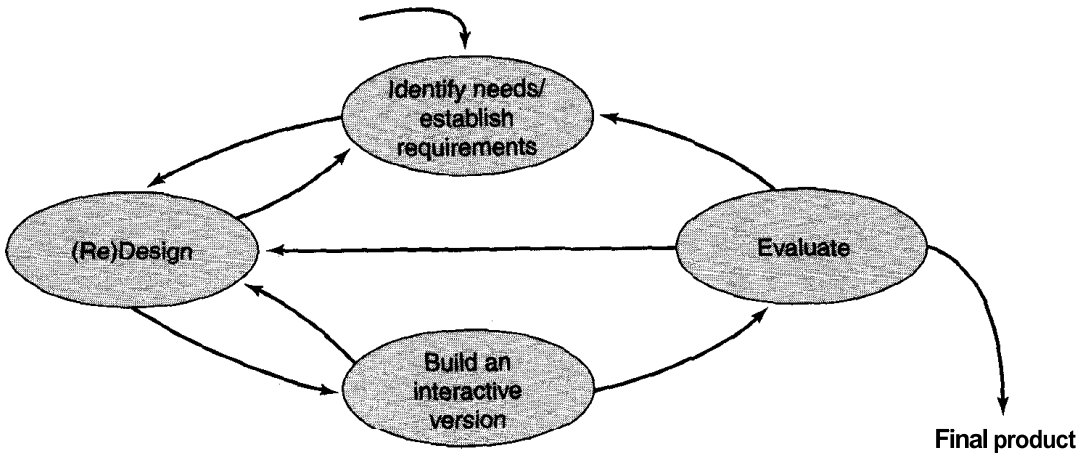


Figure 6.7 A simple interaction design model.

6.4.2 Lifecycle models in software engineering

Software engineering has spawned many lifecycle models, including the waterfall, the spiral, and rapid applications development (RAD). Before the waterfall was first proposed in 1970, there was no generally agreed approach to software development, but over the years since then, many models have been devised, reflecting in part the wide variety of approaches that can be taken to developing software. We choose to include these specific lifecycle models for two reasons: First, because they are representative of the models used in industry and they have all proved to be successful, and second, because they show how the emphasis in software development has gradually changed to include a more iterative, user-centered view.

The waterfall lifecycle model

The waterfall lifecycle was the first model generally known in software engineering and forms the basis of many lifecycles in use today. This is basically a linear model in which each step must be completed before the next step can be started (see Figure 6.8). For example, requirements analysis has to be completed before

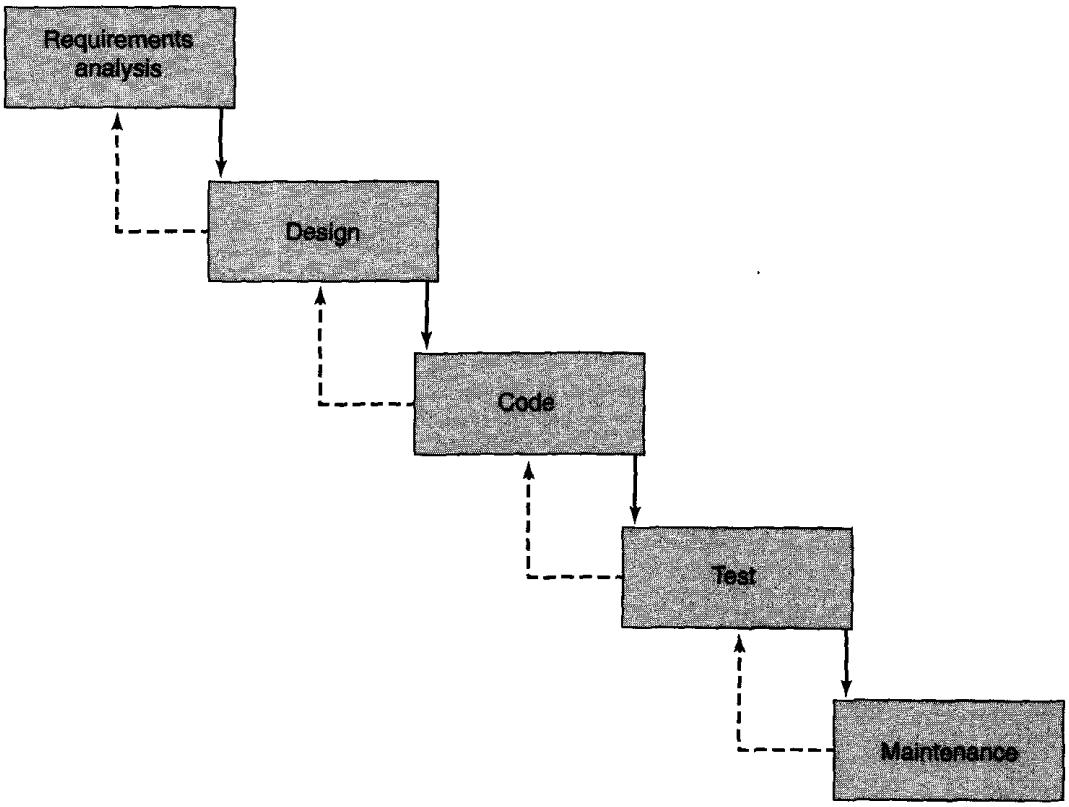


Figure 6.8 The waterfall lifecycle model of software development.

design can begin. The names given to these steps varies, as does the precise definition of each one, but basically, the lifecycle starts with some requirements analysis, moves into design, then coding, then implementation, testing, and finally maintenance. One of the main flaws with this approach is that requirements change over time, as businesses and the environment in which they operate change rapidly. This means that it does not make sense to freeze requirements for months, or maybe years, while the design and implementation are completed.

Some feedback to earlier stages was acknowledged as desirable and indeed practical soon after this lifecycle became widely used (Figure 6.8 does show some limited feedback between phases). But the idea of iteration was not embedded in the waterfall's philosophy. Some level of iteration is now incorporated in most versions of the waterfall, and review sessions among developers are commonplace. However, the opportunity to review and evaluate with ***users*** was not built into this model.

The spiral lifecycle model

For many years, the waterfall formed the basis of most software developments, but in 1988 Barry Boehm (1988) suggested the spiral model of software development (see Figure 6.9). Two features of the spiral model are immediately clear from Figure 6.9: risk analysis and prototyping. The spiral model incorporates them in an iterative framework that allows ideas and progress to be repeatedly checked and evaluated. Each iteration around the spiral may be based on a different lifecycle model and may have different activities.

In the spiral's case, it was not the need for user involvement that inspired the introduction of iteration but the need to identify and control risks. In Boehm's approach, development plans and specifications that are focused on the risks involved in developing the system drive development rather than the intended functionality, as was the case with the waterfall. Unlike the waterfall, the spiral explicitly encourages alternatives to be considered, and steps in which problems or potential problems are encountered to be re-addressed.

The spiral idea has been used by others for interactive devices (see Box 6.4). A more recent version of the spiral, called the WinWin spiral model (Boehm et al., 1998), explicitly incorporates the identification of key stakeholders and their respective "win" conditions, i.e., what will be regarded as a satisfactory outcome for each stakeholder group. A period of stakeholder negotiation to ensure a "win-win" result is included.

Rapid Applications Development (RAD)

During the 1990s the drive to focus upon users became stronger and resulted in a number of new approaches to development. The Rapid Applications Development (RAD) approach attempts to take a user-centered view and to minimize the risk caused by requirements changing during the course of the project. The ideas be-

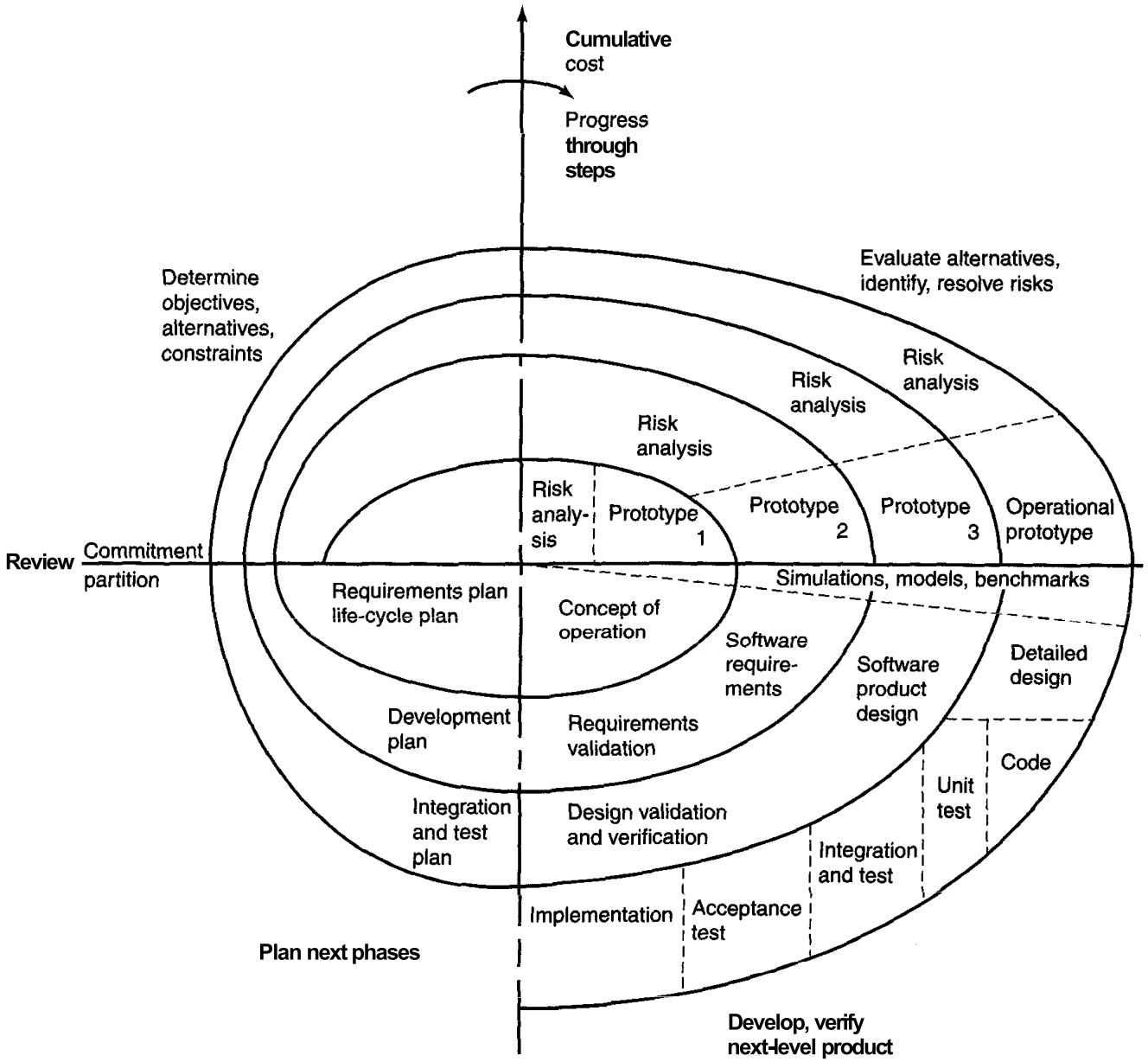


Figure 6.9 The spiral lifecycle model of software development.

hind RAD began to emerge in the early 1990s, also in response to the inappropriate nature of the linear lifecycle models based on the waterfall. Two key features of a RAD project are:

- Time-limited cycles of approximately six months, at the end of which a system or partial system must be delivered. This is called time-boxing. In effect, this breaks down a large project into many smaller projects that can deliver products incrementally, and enhances flexibility in terms of the development techniques used and the maintainability of the final system.

- JAD (Joint Application Development) workshops in which users and developers come together to thrash out the requirements of the system (Wood and Silver, 1995). These are intensive requirements-gathering sessions in which difficult issues are faced and decisions are made. Representatives from each identified stakeholder group should be involved in each workshop so that all the relevant views can be heard.

A basic RAD lifecycle has five phases (see Figure 6.10): project set-up, JAD workshops, iterative design and build, engineer and test final prototype, implementation review. The popularity of RAD has led to the emergence of an industry-standard RAD-based method called DSDM (Dynamic Systems Development Method) (Millington and Stapleton, 1995). This was developed by a non-profit-making DSDM consortium made up of a group of companies that recognized the need for some standardization in the field. The first of nine principles stated as underlying DSDM is that "active user involvement is imperative." The DSDM lifecycle is more complicated than the one we've shown here. It involves five phases: feasibility study, business study, functional model iteration, design and build iteration, and implementation. This is only a generic process and must be tailored for a particular organization.

ACTIVITY 6.5 How closely do you think the RAD lifecycle model relates to the interaction design model described in Section 6.4.1?

Comment **RAD** and **DSDM** explicitly incorporate user involvement, evaluation and iteration. User involvement, however, appears to be limited to the JAD workshop, and iteration appears to be limited to the design and build phase. The philosophy underlying the interaction design model is present, but the flexibility appears not to be. Our interaction design process would be appropriately used within the design and build stage.

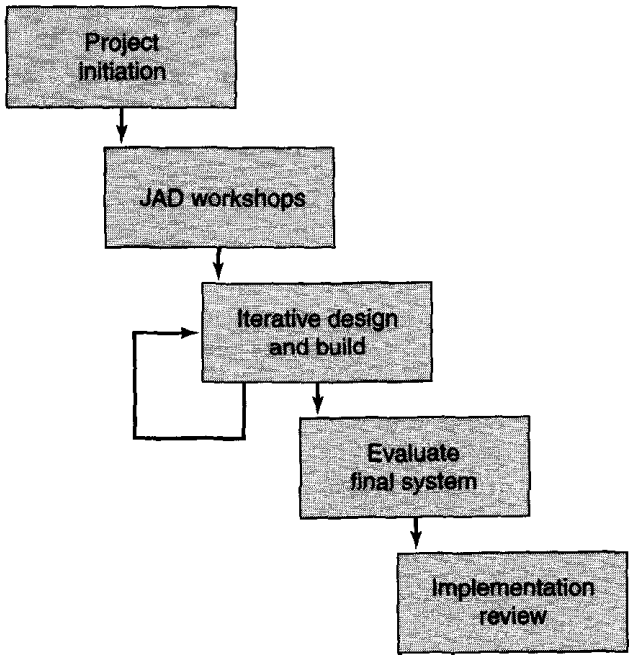


Figure 6.10 A basic **RAD** lifecycle model of software development.

BOX 6.4 A Product Design Process for Internet Appliances

Netpliance, which has moved into the market of providing Internet appliances, i.e. one-stop products that allow a user to achieve a specific Internet-based task, have adopted a user-centered approach to development based on RAD (Isensee et al., 2000). They attribute their ability to develop systems from concept to delivery in seven months to this strong iterative approach: the architecture was revised and iterated over several days; the code was developed with weekly feedback sessions from users; components were typically revised four times, but some went through 12 cycles. Their simple spiral model is shown in Figure 6.11.

The target audience for this appliance, called the i-opener, were people who did not use or own a PC and who may have been uncomfortable around computers. The designers were therefore looking to design something that would be as far away from the "traditional" PC model as possible in terms of both hardware and software. In designing the software, they abandoned the desktop metaphor of the Windows operating system and concentrated on an interface that provided good support for the user's task. For the hardware design, they needed to get away from the image of a large heavy box with lots of wires and plugs, any one of which may be faulty and cause the user problems.

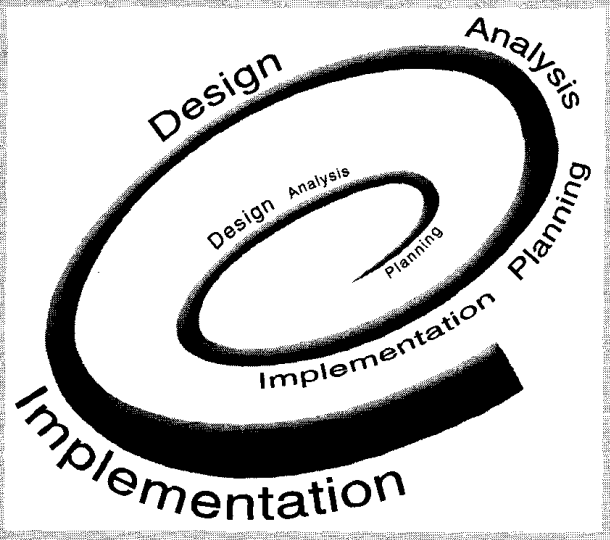


Figure 6.11 Netpliance's spiral development cycle.

The device provides three functions: sending and receiving email, categorical content, and web accessibility. That is it. There are no additional features, no complicated menus and options. The device is streamlined to perform these tasks and no more. This choice of functions was based on user studies and testing that served to identify the most frequently used functions, i.e., those that most appropriately supported the users. An example screen showing the news channel for i-opener is shown in Figure 6.12.

Identifying requirements for a new device is difficult. There is no direct experience of using a similar product, and so it is difficult to know what will be used, what will be needed, what will be frustrating, and what will be ignored. The Netpliance team started to gather information for their device by focusing on existing data about PC users: demographics, usability studies, areas of dissatisfaction, etc. They employed marketing research, focus groups, and user surveys to identify the key features of the appliance, and concentrated on delivering these fundamentals well.

The team was multidisciplinary and included hardware engineers, user interface designers, marketing specialists, test specialists, industrial designers, and visual designers. Users were involved throughout development and the whole team took an active part in the design. The interface was designed first, to meet user requirements, and then the hardware and software were developed to fit the interface. In all of this, the emphasis was on a lean development process with a minimum of documentation, early prototyping, and frequent iterations for each component. For example, the design of the hardware proceeded from sketches through pictures to physical prototypes that the users could touch, pick up, move around, and so on. To complement prototyping, the team also used usage scenarios, which are basically descriptions of the appliance's use to achieve a task. These helped developers to understand how the product could be used from a user's perspective. We will return to similar techniques in Chapter 7.

Implementation was achieved through rapid cycles of implement and test. Small usability tests were conducted throughout implementation to

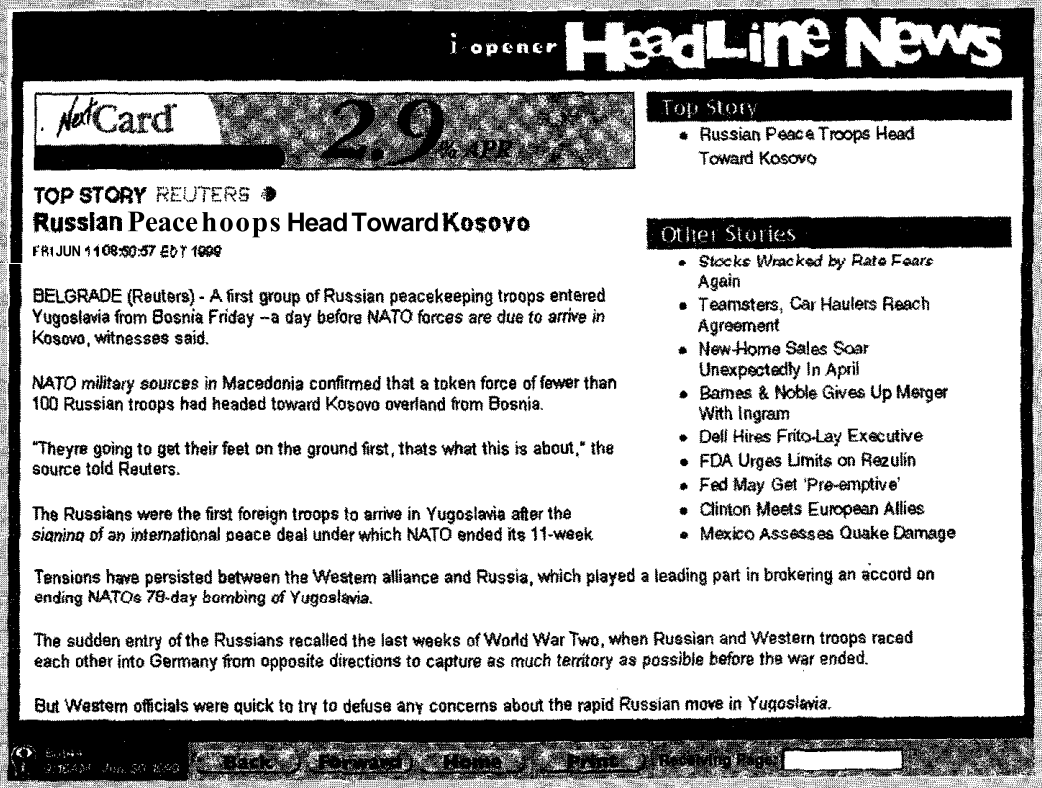


Figure 6.12 The news channel as part of the categorical content.

find and fix usability problems. Developers and their families or friends were encouraged to use the appliance so that designers could enjoy the same experience as the users (called "eating your own dogfood"!). For these field tests, the product was instrumented so that the team could monitor how often each function was used. This data helped to prioritize the development of features as the product release deadline approached.

6.4.3 Lifecycle models in HCI

Another of the traditions from which interaction design has emerged is the field of HCI (human–computer interaction). Fewer lifecycle models have arisen from this field than from software engineering and, as you would expect, they have a stronger tradition of user focus. We describe two of these here. The first one, the Star, was derived from empirical work on understanding how designers tackled HCI design problems. This represents a very flexible process with evaluation at its core. In contrast, the second one, the usability engineering lifecycle, shows a more structured approach and hails from the usability engineering tradition.

The *Star* Lifecycle Model

About the same time that those involved in software engineering were looking for alternatives to the waterfall lifecycle, so too were people involved in HCI looking for alternative ways to support the design of interfaces. In 1989, the Star lifecycle

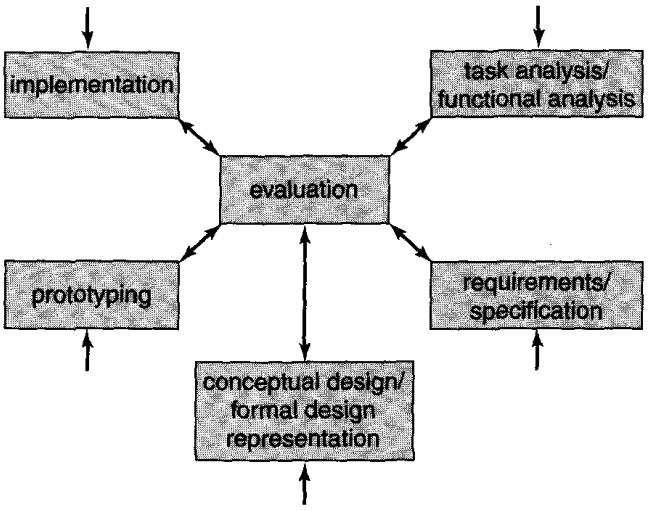


Figure 6.13 The Star lifecycle model.

model was proposed by Hartson and Hix (1989) (see Figure 6.13). This emerged from some empirical work they did looking at how interface designers went about their work. They identified two different modes of activity: analytic mode and synthetic mode. The former is characterized by such notions as top-down, organizing, judicial, and formal, working from the systems view towards the user's view; the latter is characterized by such notions as bottom-up, free-thinking, creative and ad *hoc,* working from the user's view towards the systems view. Interface designers move from one mode to another when designing. A similar behavior has been observed in software designers (Guindon, 1990).

Unlike the lifecycle models introduced above, the Star lifecycle does not specify any ordering of activities. In fact, the activities are highly interconnected: you can move from any activity to any other, provided you first go through the evaluation activity. This reflects the findings of the empirical studies. Evaluation is central to this model, and whenever an activity is completed, its result(s) must be evaluated. So a project may start with requirements gathering, or it may start with evaluating an existing situation, or by analyzing existing tasks, and so on.

ACTIVITY 6.6 The Star lifecycle model has not been used widely and successfully for large projects in industry. Consider the benefits of lifecycle models introduced above and suggest why this may be.

Comment One reason may be that the Star lifecycle model is extremely flexible. This may be how designers work in practice, but as we commented above, lifecycle models are popular because "they allow developers, and particularly managers, to get an overall view of the development effort so that progress can be tracked, deliverables specified, resources allocated, targets set, and so on." With a model as flexible as the Star lifecycle, it is difficult to control these issues without substantially changing the model itself.

The Usability Engineering Lifecycle

The Usability Engineering Lifecycle was proposed by Deborah Mayhew in 1999 (Mayhew, 1999). Many people have written about usability engineering, and as

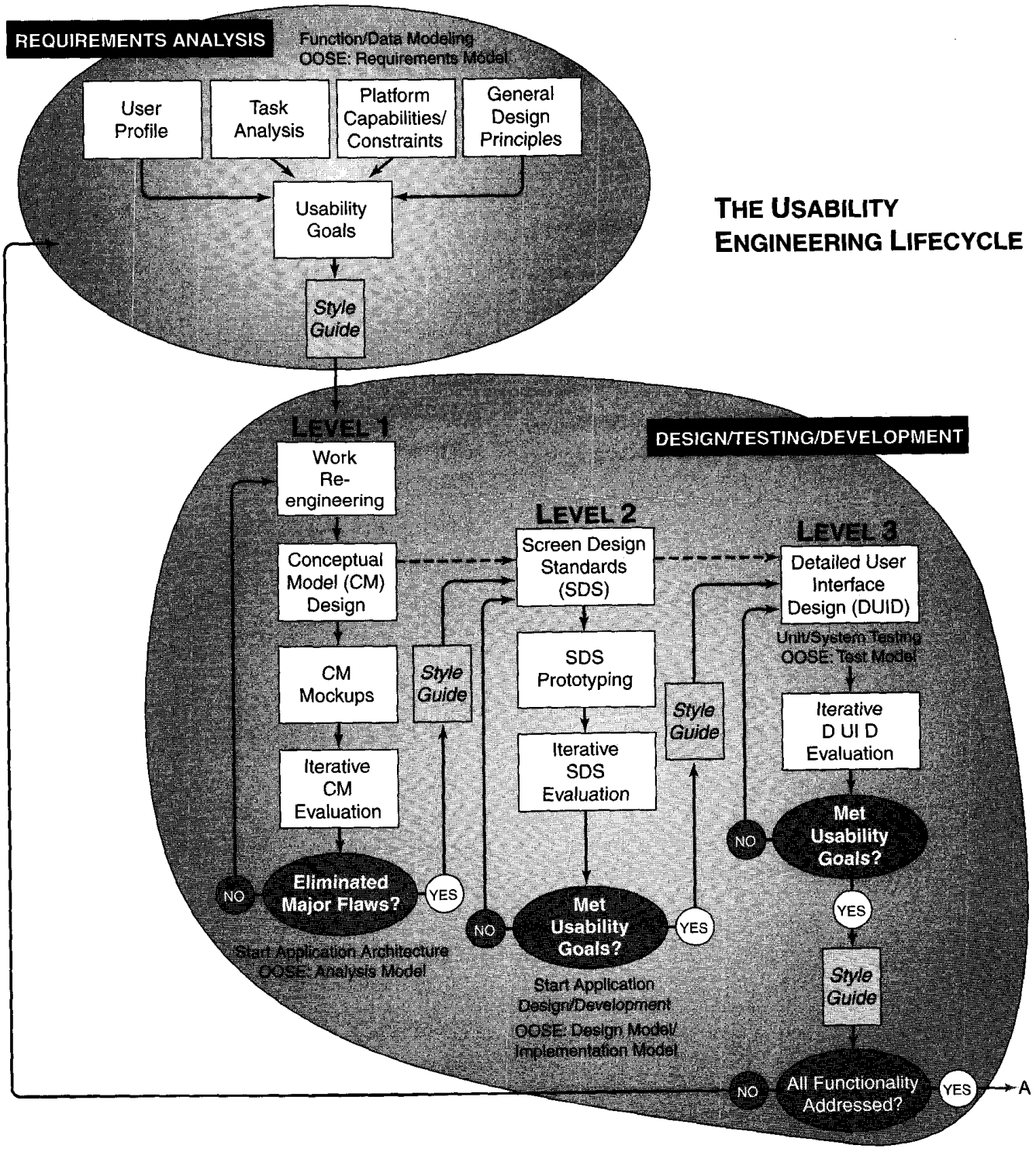


Figure 6.14 The Usability Engineering Lifecycle.

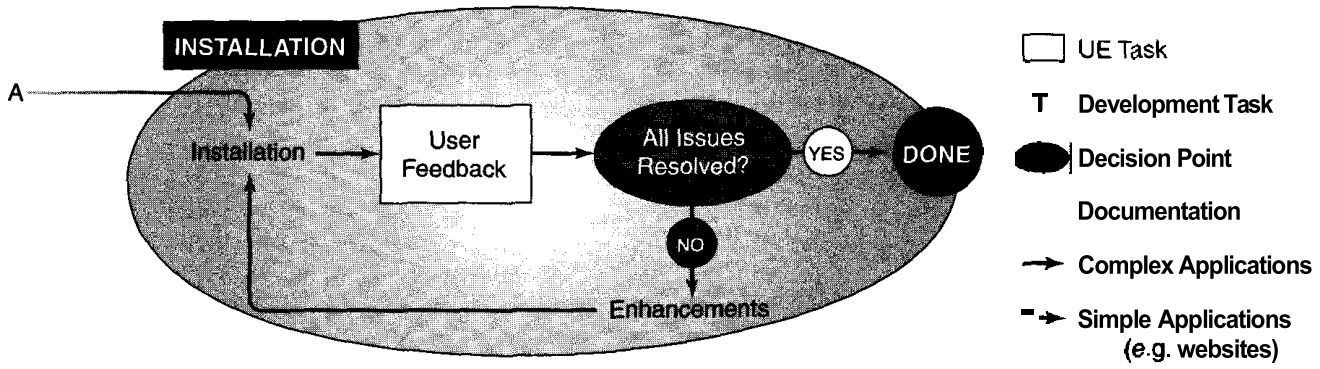


Figure 6.14 *(continued).*

Mayhew herself says, "I did not invent the concept of a Usability Engineering Lifecycle. Nor did I invent any of the Usability Engineering tasks included in the lifecycle . . . .". However, what her lifecycle does provide is a holistic view of usability engineering and a detailed description of how to perform usability tasks, and it specifies how usability tasks can be integrated into traditional software development lifecycles. It is therefore particularly helpful for those with little or no expertise in usability to see how the tasks may be performed alongside more traditional software engineering activities. For example, Mayhew has linked the stages with a general development approach (rapid prototyping) and a specific method (object-oriented software engineering (OOSE, Jacobson et al, 1992)) that have arisen from software engineering.

The lifecycle itself has essentially three tasks: requirements analysis, design/testing/development, and installation, with the middle stage being the largest and involving many subtasks (see Figure 6.14). Note the production of a set of **usability** goals in the first task. Mayhew suggests that these goals be captured in a style guide that is then used throughout the project to help ensure that the usability goals are adhered to.

This lifecycle follows a similar thread to our interaction design model but includes considerably more detail. It includes stages of identifying requirements, designing, evaluating, and building prototypes. It also explicitly includes the style guide as a mechanism for capturing and disseminating the usability goals of the project. Recognizing that some projects will not require the level of structure presented in the full lifecycle, Mayhew suggests that some substeps can be skipped if they are unnecessarily complex for the system being developed.

ACTIVITY 6.7 Study the usability engineering lifecycle and identify how this model differs from our interaction design model described in Section 6.4.1, in terms of the iterations it supports.

Comment One of the main differences between Mayhew's model and ours is that in the former the iteration between design and evaluation is contained within the second phase. Iteration between the design/test/development phase and the requirements analysis phase occurs only after the conceptual model and the detailed designs have been developed, prototyped, and

evaluated one at a time. Our version models a return to the activity of identifying needs and establishing requirements after evaluating any element of the design.

Assignment

Nowadays, timepieces (such as clocks, wristwatches etc) have a variety of functions. They not only tell the time and date but they can speak to you, remind you when it's time to do something, and provide a light in the dark, among other things. Mostly, the interface for these devices, however, shows the time in one of two basic ways: as a digital number such as 23:40 or through an analog display with two or three hands—one to represent the hour, one for the minutes, and one for the seconds.

In this assignment, we want you to design an innovative timepiece for your own use. This could be in the form of a wristwatch, a mantelpiece clock, an electronic clock, or any other kind of clock you fancy. Your goal is to be inventive and exploratory. We have broken this assignment down into the following steps to make it clearer:

(a) Think about the interactive product you are designing: what do you want it to do for you? Find 3–5 potential users and ask them what they would want. Write a list of requirements for the clock, together with some usability criteria based on the definition of usability used in Chapter 1.

(b) Look around for similar devices and seek out other sources of inspiration that you might find helpful. Make a note of any findings that are interesting, useful or insightful.

(c) Sketch out some initial designs for the clock. Try to develop at least two distinct alternatives that both meet your set of requirements.

(d) Evaluate the two designs, using your usability criteria and by role playing an interaction with your sketches. Involve potential users in the evaluation, if possible. Does it do what you want? Is the time or other information being displayed always clear?

Design is iterative, so you may want to return to earlier elements of the process before you choose one of your alternatives.

Once you have a design with which you are satisfied, you can send it to us and we shall post a representative sample of those we receive to our website. Details of how to format your submission are available from our website.

Summary

In this chapter, we have looked at the process of interaction design, i.e., what activities are required in order to design an interactive product, and how lifecycle models show the relationships between these activities. A simple interaction design model consisting of four activities was introduced and issues surrounding the identification of users, generating alternative designs, and evaluating designs were discussed. Some lifecycle models from software engineering and HCI were introduced.

Key points

- The interaction design process consists of four basic activities: identifying needs and establishing requirements, developing alternative designs that meet those requirements, building interactive versions of the designs so that they can be communicated and assessed, and evaluating them.

- Key characteristics of the interaction design process are explicit incorporation of user involvement, iteration, and specific usability criteria.
- Before you can begin to establish requirements, you must understand who the users are and what their goals are in using the device.
- Looking at others' designs provides useful inspiration and encourages designers to consider alternative design solutions, which is key to effectivedesign.
- Usability criteria, technical feasibility, and users' feedback on prototypes can all be used to choose among alternatives.
- Prototypingis a useful technique for facilitating user feedback on designs at all stages.
- Lifecycle models show how development activities relate to one another.
- The interaction design process is complementary to lifecycle models from other fields.

Further reading

RUDISILL, M., LEWIS, C., POLSON, P. B., AND MCKAY, T. D. (1995) (eds.) ***Human-Computer Interface Design: Success Stories, Emerging Methods, Real-World Context.*** San Francisco: Morgan Kaufmann. This collection of papers describes the application of different approaches to interface design. Included here is an account of the Xerox Star development, some advice on how to choose among methods, and some practical examples of real-world developments.

BERGMAN, ERIC (2000) (ed.) ***Information Appliances and Beyond.*** San Francisco: Morgan Kaufmann. This book is an edited collection of papers which report on the experience of designing and building a variety of 'information appliances', i.e., purpose-built computer-based products which perform a specific task. For example, the Palm Pilot, mobile telephones, a vehicle navigation system, and interactive toys for children.

MAYHEW, DEBORAH J. (1999) ***The Usability Engineering Lifecycle.*** San Francisco: Morgan Kaufmann. This is a very practical book about product user interface design. It explains how to perform usability tasks throughout development and provides useful examples along the way to illustrate the techniques. It links in with two software development based methods: rapid prototyping and object-oriented software engineering.

SOMMERVILLE, IAN (2001) *Software* ***Engineering*** (6th edition). Harlow, UK: Addison-Wesley. If you are interested in pursuing the software engineering aspects of the lifecycle models section, then this book provides a useful overview of the main models and their purpose.

NIELSEN, JAKOB (1993) ***Usability Engineering.*** San Francisco: Morgan Kaufmann. This is a seminal book on usability engineering. If you want to find out more about the philosophy, intent, history, or pragmatics of usability engineering, then this is a good place to start.