Chapter 14
Summary: Putting It All Together

Abstract This chapter recaps some of the many things that you have learned about users in terms of their anthropometric, behavioral, cognitive, and social aspects. You have been provided with a lot of information, so we describe a number of different possible ways you can organize it. One way to organize and apply the information is with user models. These models span the range from implicit descriptive models, such as guidelines, through to explicit information processing models, which can be executed to produce behavior and predict performance. Another way is to organize the information based on how to use it. So we finish by looking at one system development process model—the Risk-Driven Incremental Commitment Model—as an example of how you can integrate knowledge about users into the system development life cycle. Failure to consider the users and their tasks during development leads to increased system development risk.

14.1 Introduction

Human centered-design is about putting humans at the center of system design. If we want to design systems that are both useful and usable, we need to understand humans—users, in particular—and this is not a trivial undertaking. If we do not take appropriate account of the users there is an increased risk that the project will be a failure: the users could refuse to accept the system if it does not fit into and support the way they work, or the users may end up wrestling with the system trying to make it behave as they expect it to, possibly with fatal consequences (e.g., Baxter et al. 2007), or it could just not be fully adopted by users.

Developing systems is an inherently interdisciplinary practice. The knowledge that we have presented here should make software and system engineers at least aware of the sorts of issues that human factors engineers routinely discuss. In this book we have tried to highlight the capabilities and limitations of humans but, really, we have just scratched the surface. We are not expecting software engineers to become human factors experts as a result of reading this book. We hope,

DOI: 10.1007/978-1-4471-5134-0_14, © Springer-Verlag London 2014

however, that it will enable them to communicate with human factors engineers, and develop a shared understanding of the issues as they affect system development and use.

We conclude our discussions with a brief summary of what we have presented in this book. You will want to organize it yourself in some meaningful way to be able to make effective use of all the information about users that we have presented. How you decide to organize it will partly depend on how you intend to use it. You might find the metaphors in Chap. 12, those of the Gulfs and Cognitive Dimensions, to be useful. There are also methods and techniques that can help you. A few of these have been implemented as computer programs that are fairly easy to use, and encapsulate summaries of user behavior that can be applied in a range of ways. These models are just one way of applying what we know about humans to system design; they can also be used to help teach designers and developers about users as a sharable representation.

We finish by describing how you can take forward what you have learned here and incorporate it into system development. We would contend that you should always apply this knowledge, because a lack of knowledge about users nearly always constitutes a risk to system development. We illustrate this argument using an extended version of the Risk-Driven Incremental Commitment Model (Pew and Mavor2007).

14.2 Organizing What We Have Learnt About Users

If your aim is to build systems that are both useful and usable then the best way forward in many cases is to focus on particular people doing particular tasks in particular contexts. The knowledge that we have provided you will help. Whether a system can be described as usable or not depends on factors such as the shape and size of the users (anthropometric factors), external body functioning and simple sensory-motor concerns and motivation (behavioral factors), internal mental functioning (cognitive factors), and external mental functioning (social and organizational factors). It therefore makes sense to consider organizing your knowledge about these different factors in terms of the types of user characteristics that we introduced early in the book: Anthropometric; Behavioral; Cognitive; and Social—the ABCS. You should be aware that there are other factors too, which we have not addressed in this summary such as legal liability, and physiological factors which will often also be important.

14.2.1 Anthropometrics

Users vary. One obvious way they can vary is in physical size, as Chap. 3 noted. Different people of the same age and gender may have different heights and different weights, for example. Users also change over time as they develop and age. The physical attributes of the user will affect how they use a particular artifact, so you need to consider aspects such as whether they can reach the controls, whether they can operate the levers, whether they can push buttons, and so on.

The way that people use artifacts can also affect the well-being of the user. Upper limb disorders, for example, can arise from having to carry out the same task repeatedly over extended periods of time, and from the user failing to adopt the correct posture.

Anthropometrics is an important consideration in interfaces where touch) plays a central role. Probably the most obvious and widespread examples are smart-phones and tablet computers. In addition, it is important when thinking about keyboards: conventional keyboards are still widely used with most personal computers, and many people use their thumbs to type on the keypads of some cell phones, for example.

14.2.2 Behavior

The user's behavioral characteristics are mostly related to perception, and the most important of these are vision and audition, as noted in Chap. 4. Users will also differ in terms of their perceptual capabilities. As people get older, for example, their vision and hearing often diminish and there are people who have permanently impaired vision or hearing.

When you are designing systems it is therefore important to realize that all users will not always behave in exactly the same way. Their performance will vary across a range of behavior which is approximately normally distributed, as shown on the left in Fig. 14.1. In those situations where the distribution cannot be two tailed, such as where reaction times (which cannot be negative!) are being measured, performance more closely approximates a gamma distribution, as shown on the right in Fig. 14.1.

Understanding the way that people perceive things is important, because this understanding can be used to create artifacts that more closely match the way that users behave. Knowing about red–green color deficiency, for example, is important because it will influence the way that colored items are used in an interface. At a higher level, knowing the Gestalt laws that describe the way people perceive groups of objects (Chap. 4) can be used to help you work out the layout of objects in an interface.

When it comes to designing audio outputs, knowing the way that hearing works can help you determine what sort of sounds to use for alarms, for example. We also know that people are poor at localizing sound, but it is easier to localize high pitch sounds than low pitch ones, and that speech conveys more information than sounds.

All users, from novices to experts, make errors and in some cases make the same errors as noted in Chap. 10. One of the main differences between novices and

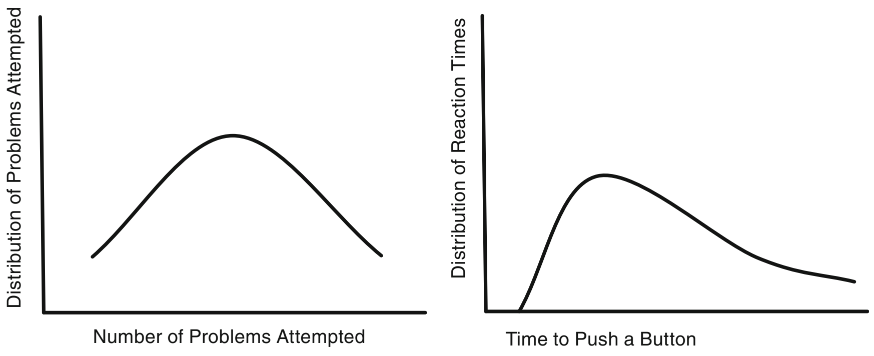

Fig. 14.1 A normal curve (*left*) [Strictly speaking, number of problems cannot go below 0 either, but for most practical examples, this distribution will be well represented by a normal curve when 0 is more than three standard deviations from the mean (the peak).] and a gamma curve (*right*) showing typical distributions for tasks attempted and response times for a task. These curves illustrate that users will have a distribution of behavior, and not always perform a task or have knowledge in a single way

experts is that expert users can often (but not always) detect and correct errors before any adverse consequences arise. When you design systems, you need to remember that all users will make errors. Some of these errors will be preventable, but others may not. You will therefore need to decide which errors you will prevent (using risk assessment, for example). For the other errors, you will need to include support to help the user spot errors and, in particular, to support the ability to correct errors, ideally before any adverse consequences arise.

Where the users have physical limitations, however, or there are contextual limitations (low lightinglevels, for example), these limitations will constrain what the users can and cannot do.

14.2.3 Cognition

Users' cognition is limited, at both the tactical level (e.g., limited working memoryand some difficulties in storing and retrieving items from memory) and at the strategic level (e.g., how to decide which are the important and long term issues), as noted in Chaps. 5–7. In some cases, the effects of these limitations can be ameliorated through learning, or through social processes, particularly where skills are pooled to perform particular tasks.

Attracting, managing, and maintaining the user's attention are all important. Users have limited attentional resources. This means that in busy contexts, where lots of things are happening or several tasks have to be performed simultaneously, they are unlikely to be able to attend to all of them. In other words, they will find it difficult to consciously control how they perform all the tasks. With practice, users can learn to perform some tasks with little or no conscious control (i.e., with little or no attention). The classic example that is usually quoted is that of riding a bike: when you first start out, you try to think of lots of things at once—keeping your balance, remembering to pedal, keeping the handlebars straight, being aware of the traffic around you, and so on—but as you learn how to do the task, you will focus more on just keeping the bike going in a straight line, and maintaining awareness of the traffic around you. We know that people learn in various ways, so you should design your system in such a way that it facilitates learning.

It is important to know your users, and to understand the language that they use. By doing so, you will be able to develop systems that they can more readily understand and use. The interaction between the user and their computer-based system can be considered as a form of conversation, which means that it can be optimized using Grice's maxims (Chap. 7). It is particularly important to know something about how people read and the ways that they seek out information (mostly through searching), because these are two of the most common user activities. You should design systems that present information in ways that the users can easily read, understand, and find. This means that you need to think about presentation (font choices, for example) as well as content issues.

Users have mental models of how things work, and the way they interact with an artifact is usually based on the (most) relevant mental model. These models develop over time as the user's knowledge and experience of using the artifact increases. As a system designer you should try to help the user to develop the right model of how your system operates. The mental model will help the users decide which action to take next. When they cannot see which action to perform, they will normally engage in problem solving behavior, which may involve reasoning from first principles, using trial and error, for example. On the basis of their problem solving, users will then make decisions. Decision making is usually sub-optimal because people do not take into account all of the relevant information that is available to them, and because their decision making processes are often biased in known ways. As a designer you can try to help users by making relevant information salient and by counteracting the known biases where possible.

The use of mental models also applies to users themselves. Across the range of behavior types, there are very few aspects of their behavior that users know about intrinsically. They can see their size, and this they know something about. Yet when it comes to choosing a chair, they have to sit in it to know if it fits. Many users do not sit in the best posture, and so we might note that many users do not know all that they could about themselves anthropometrically.

Many users do not know about their fovea, and most other aspects of perception are not available to consciousness—it is impenetrable. The results of vision are available when objects are attended to, but the parts that are not attended to are ignored or filled in. For example, most users will not know much about the blind spot on their retina, let alone see it. Most of perception is impenetrable. So, what they know about perception is mostly folk psychology.

Users (and untrained designers) think they know how they think. This is often misguided. While they are thinking about how they think, they are busy thinking. Imagine creating a computer that stored every action it took—but how can it do this? It would have to record actions while it was doing them. Users are basically like this as well. It is very hard to remember what you are doing when you are busy doing it. Ericsson and Simon (1993) laid out a fairly good theory of when users can talk aloud about what they are doing. It is even more difficult to talk about what you have done and be completely accurate. That said, sometimes the only data we can afford to gather is how users think they think, and sometimes users will choose systems based on such judgments, so we will often have to work with these judgements. You might wish to be more skeptical in the future about users' (and untrained designers') reports about how they do a task or whether they learn or not.

14.2.4 Social

Most tasks are carried out by teams of people interacting with one another. Sometimes they will be working directly with co-located people, but in other cases they may be working distally as a team. Individual users have limitations on how well they work together in teams: some people are natural leaders, whilst some are natural followers.

Chapters 8 and 9 note social factors that influence users, including distributed responsibility, and social influence on teams. For example, users (and system developers) will often misjudge social relationships. They will often send out one blanket email to ten people asking for one of them to volunteer, rather than sending out ten individual emails. The sorts of variations in performance that you see across users can sometimes be attributed to the context they are working in, as well as their own capabilities and limitations. If you put the same user in the same task and physical situation but vary the environmental conditions by increasing the temperature, or reducing the lightinglevels, for example, this may affect their performance.

14.2.5 The Role of Tasks and Environments

Knowing users' tasks will help design systems to perform these tasks in numerous ways as presented in Chap. 11. There are a wide range of uses and applications, and of ways to represent users' tasks and activities. Good designers can choose appropriately based on what is needed for a given design project.

Users' tasks are not always directly understandable by designers using their own intuitions, but there are ways to find what tasks users do and what tasks they want to do. However, there will still be surprises because it is difficult to know everything, particularly for new tasks and unexpected uses.

It is also important to understand the environment in which users perform their tasks. Usability studies help to provide a way to understand the tasks, the users, the environments, and how they interact.

14.2.6 Summary

Users can be viewed as having some aspects that are more uniform and some that are more unique. There are some communalities across levels that we can point out:

1. The ABCS note some of these aspects where users have shared communalities in how they think, interact with the world, and interact with each other; in some ways users are alike. So, when designing systems, you should not despair that all users are different.
2. There are also differences between users. There are simple differences in simple capabilities, such as different input and output speeds and capabilities. In more complex capabilities, like how to do a task and mental models of systems, users can vary widely based on previous experience and practice. In cases of goals and cultural beliefs they can vary the most. So, when designing, you need to keep in mind that, for some aspects of design, users can differ.
3. Users have limited capabilities. Across the chapters, limitations on how fast users can move, how fast and how they can think, and their abilities to produce perfect, error-free performance were noted. Most importantly, user limitations can be avoided by better design that does not make incorrect assumptions about users. Many of the bad web site design contests and bad and dangerous interface galleries are filled with designs that assumed something incorrect about the users' abilities or interests. So, when designing, you need to design for what people can do.
4. Users do not always know themselves. You can ask them, but they will sometimes not provide very accurate descriptions. Designers are also a type of user, and they suffer the same limitations about knowing users and themselves. So, when designing, do ask them but also think more broadly about users and their context. In addition, try to study the user as a domain. Have a theory of how they perform tasks, test this theory with data, and expand the theory as you learn and read more.
5. Systems can be helped in a variety of ways because of users. Users can learn, can then find new strategies, and can help each other. So, when you design, keep these less obvious changes in mind.

14.3 Models of Users

For the purpose of engineering design, it would be useful to model the human part of the overall system in a more formalized way, in the same way that engineering design specifications can be used to present a relatively clear and precise model for implementation. The model of the user would serve as a shared representation that could be used to support system design. From the human factors engineer's point of view, a user model captures the capabilities and limitations on user performance; from the software engineer's point of view, it would be used to illustrate how the system could perform when operated by real users.

Models of users therefore serve as a summary repository of our knowledge of users. It is important that this knowledge be captured in one place because it can lead to emergent behaviors where there are interactions between the different characteristics of users. The behavior of the models should be constrained in the same sorts of ways that human behavior is constrained (memory limitations, and so on).

We have seen throughout this book that humans are less predictable, consistent, and deterministic than computers. Defining a general, formal model of the human (as part of the broader socio-technical system) is not currently possible. Instead, a number of fragmentary and incomplete models of human information processing behavior have been proposed by cognitive psychologists and reinforced by empirical exploration. These models can be used to make predictions but do not provide details about how the system should be designed. Although the models are good at generating first-order effects, at lower levels of analysis their limitations and inconsistencies become apparent. The models are therefore useful at predicting gross behavior, such as error-free, expert behavior on unit tasks.

The most accurate, detailed models have generally been developed for those aspects of human performance that are easiest to test. Thus, models of the characteristics of the senses are well established (particularly vision and hearing), whereas there are very few models of some of the more intricate aspects of cognition that can only be indirectly observed and are less well understood.

14.3.1 Unified Theories of Cognition

There have been several attempts to integrate all that we know about human behavior into a single, unified theory of cognition (Newell 1990). The latest attempts have been realized as cognitive architectures, although they might be more properly described as human performance architectures because they deal with more than just the cognitive aspects of human performance. These cognitive architectures typically take the form of a computer programming language with special capabilities and limitations representing a (small) subset of the known capabilities and limitations of humans. Progress has been relatively slow, but the cognitive architecture developers are working towards providing a single coherent theory that ultimately accounts for—and predicts—human performance on all tasks.

A unified theory of cognition effectively forms a single repository of useful user-related information. This can be used in three main ways:

1. To help you remember theories and facts about users. You can use the schematic of the Model Human Processor (MHP, described below), ACT-R (also described

below), or Soar (Laird 2012), for example, to work your way through their hypothesized modules to understand how behavior will be generated. At this level of abstraction, there is a fair amount of agreement between the theories in that they all include modules for input, memory, cognition, and output.
2. To summarize user behavior. In time, architectures may be used more often during design as a way to conveniently capture theories of user behavior, and as a teaching aid to help designers understand users (Pew and Mavor 2007).
3. To apply what we know about users. In some cases, the models only offer fairly crude approximations of users, but they have been useful for populating computer games and military simulations. In other cases, they are being used to test interfaces and make predictions for designs (for a review, see any of these reports: Booher and Minninger 2003; Pew and Mavor 1998, 2007; Ritter et al. 2003).

14.3.2 Types of User Models

There are now more than 100 cognitive (and user) architectures if you include variants and different versions (Morrison 2003; Pew and Mavor1998; Ritter et al. 2003). They can be categorized into four types:

1. Implicit descriptive models. When you look at a car, for example, you can imagine some of the assumptions the designers have made about the driver. These assumptions are captured in an implicit model of the driver in which vision takes place in the top part of the body, and another part of the body operates the controls on the floor.
2. Explicit declarative models. These models describe the components or a structure in a system, but neither describe the mechanisms nor perform the task.
3. Explicit process models. The mechanisms and the way they operate are described, but the models, while perhaps supported by software, do not process information.
4. Explicit information processing models. These models include a full information processing architecture that produces behavior, and can predict performance times and the information processing steps that will be performed and their results.

14.3.2.1 Implicit Descriptive Models

Implicit user models appear in many systems, and some people would claim that all systems include a model of the user. For example, chairs assume a height of the user, and many file systems assume users can read and write English. Taken together, the set of assumptions used by the designer to create an artifact is an implicit model of the user, but it may be a particularly impoverished, incomplete, or incorrect model. There are certainly several tools and approaches that include models of users where the model is clear enough to talk about, but is not explicitly represented and may not be visible (or even known) to most people. Guidelines and web accessibility tools, for example, show that models can be useful even when they are not explicit, examinable, or even very modifiable.

Guidelines attempt to describe how to build a system to support users. The guidelines reflect an implicit model of the user (which may be more explicitly specified by the guideline developers). Guidelines that give advice about visual layout, for example, may assume that the users have normal (20/20) vision which, clearly, all users do not.

It would be difficult for designers to use these models and to manually apply tests against the guidelines. Where the guidelines have been encapsulated in software tools they are relatively easy to apply. Some of these tools only indicate compliance (or otherwise) with the guidelines, however, and do not explain why particular features are undesirable.

Where guidelines are implemented in software, for example, to test the accessibility of a web site, an implicit user model is employed to evaluate the interface against those guidelines. Tools like Bobby and Truwex (search online or see the book's web site), for example, assess the accessibility of web sites against the Web Content Accessibility Guidelines (WCAG, www.w3.org/WAI/intro/wcag.php). For a more extensive discussion of automated tools for evaluating interfaces see Ivory and Hearst's (2001) review that divides the tools into various categories and assesses their potential impact.

14.3.2.2 Explicit Descriptive Models

Explicit descriptive models describe the process and mechanisms that make up user behavior. These models include task analysis. Examples include models built with the KLM and with GOMS (Chap. 11) because they simply describe the behavior (a trace of the behavior) without describing in detail the inner workings of how the behavior is implemented in cognitive and other mechanisms.

These models can be very useful in system development. Examples of tools to help create such models include Interacting Cognitive Subsystems (ICS, Barnard 1987) and the Improved Performance Research Integration Tool (IMPRINT, Booher and Minninger 2003). IMPRINT has probably had the largest impact. It describes the tasks that users have to perform and how long each task should take. The resulting model is then used to predict how performance is degraded by fatigue and how many users are required to perform the set of tasks.

There are also tools that try to replicate the time course of the interaction, sometimes interacting with a simulation of the external world. The GOMS Language Evaluation and Analysis tool (GLEAN), for example, encapsulates the GOMS task analysis methodology (Kieras 1999; Kieras et al. 1995) in a way that makes it easier to use and apply GOMS models. This approach really sits halfway between descriptive models and process models that perform the task by processing information.

14.3.2.3 Explicit Process Models: Model Human Processor

Card et al. (1980, 1983) believed applying information processing psychology should be based on task analysis, calculation, and approximation. Their Model Human Processor (MHP) offers an early, simple, integrated description of psychological knowledge about error-free human performance relevant to HCI and system design. It was one of the first attempts to get away from the proliferation of descriptions developed to account for different psychological observations, and to provide a unified description of users. It started to create a quantitative methodology including average times. The MHP was an oversimplification, but it did provide a kind of prototype model that could be used as the basis for discussions.

The MHP was made up of a set of memories and processors, together with a set of principles, including the "principles of operation," which describe how the components functioned together. There were three interacting subsystems: the perceptual system, the motor system, and the cognitive system, each of which had their own memories and processors. These are shown schematically in Fig. 14.2. A detailed description of how the MHP would perform a task is achieved using either a KLM analysis or a GOMS analysis, described in Chap. 11. Examples of MHP analyses are shown in Table 14.1, and two more are on the book's web site.

Long term memory essentially does not decay ($\delta = \infty$) and has infinite capacity ($\mu = \infty$). Working memory has a visual store and an auditory store. Memories in both stores decay, but at different rates, and the sizes are different. Working memory outside the stores has a capacity limitation. The perceptual processor consists of sensors and associated buffer memories, the most important being a Visual Image Store, and an Auditory Image Store to hold the output of the sensory system while it is being symbolically coded. The cognitive processor receives symbolically coded information from the sensory image store into its working memory, and uses information previously stored in long-term memory to decide how to respond. The motor processor carries out the selected response.

Time predictions are generated by analyzing a task into the constituent operations that are executed by the subsystems. Then average times are associated with these operations based on the selected band of performance: Fastman, Slowman, and Middleman, which allows predictions to be made along the central and extreme points of the behavioral continuum of fast to slow users.

The MHP assumes highly idealized behavior (e.g., a single strategy to solve a task), and has trouble representing errors. The latter is important. Errors in text editing, for example, have been shown to account for 35% of expert performance time and 80% of the variability in that time (Landauer 1987, p. 151). Although the representations of the perceptual, cognitive, and motor subsystems were weak, the MHP did demonstrate the feasibility of the general idea and inspired later work on information processing cognitive architectures realized as computer programs, such as Soar (Newell 1990) and ACT-R (Anderson 2007).

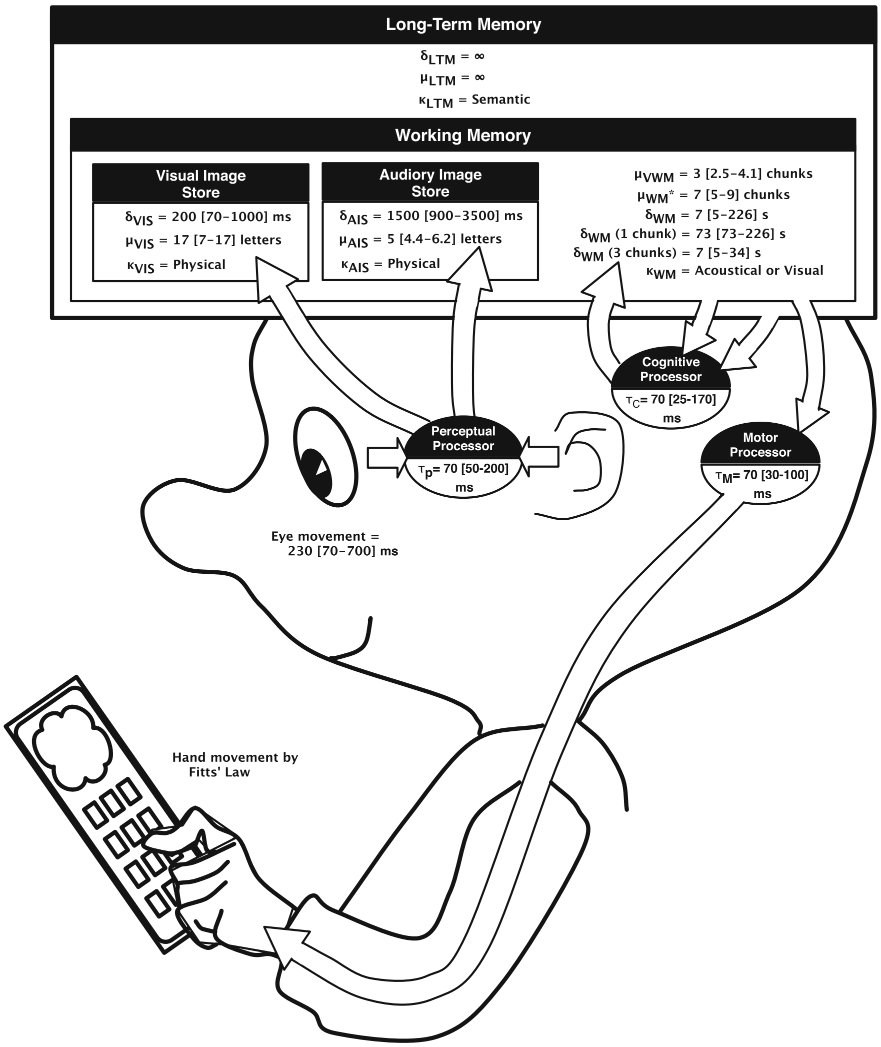

Fig. 14.2 A diagram of the model human processor (adapted from Card et al. 1983). δ indicates a half-life decay rate; μ is a capacity; κ is the modality of the storage; τ is the time to do something

14.3.2.4 Explicit Information Processing Models: ACT-R

There is now a range of cognitive architectures that take as input a descriptive task analysis that looks a lot like a computer program, and then simulate the cognitive aspects of task performance, often with limitations simulated from vision and motor performance. One of the most widely used architectures is ACT-R; others include EPIC (Kieras et al. 1997) and Soar (Laird 2012; Ritter 2003; Ritter and Bibby 2008).

Table 14.1 Example MHP analysis

MHP: example 1: motor skills, typing behavior
A manufacturer is considering whether to use an alphabetic keyboard on his handheld point of sale (PoS) system. Among several factors influencing his decision is the question of whether experienced users will find the keyboard slower for touch typing than the standard Sholes (QWERTY) keyboard arrangement. What is the relative typing speed for expert users on the two keyboards?
Typing rate = 152/ks (72 words/min)
Typing rate (alphabetic) = 164 ms/key (66.5 words/min)
MHP: example 2: cognitive, working memory
A programmer is told verbally the one-syllable file names of a dozen files to load into his programming system. Assuming all the names are arbitrary, in which order should the programmer write down the names so that he remembers the greatest number of them (has to ask for the fewest number to be repeated)?
The fact that there are 12 arbitrary file names means the programmer has to remember 12 chunks (assuming one chunk/name), which is larger than the storage capacity of working memory, so some of the file names will be forgotten. The act of trying to recall the file names will add new items to working memory, interfering with the previous names. The items likely to be in working memory but not yet in long-term memory are those from the end of the list. If the task is to recall the names from the end of the list first, he can snatch some of these from working memory before they are displaced. The probability of recalling the first names will not be affected because if they are available, they are in long-term memory. Thus the programmer should recall the last names first and then the others but will forget some

ACT-R has been continually evolved and updated—the latest version is available at act.psy.cmu.edu. The structure of the latest version of ACT-R (see Fig. 14.3) is somewhat similar to that of the MHP. The figure shows the modules and mechanisms of cognition. It also attempts to show a mapping between the mechanisms and the areas of the brain responsible for creating the behavior. This correspondence is not perfect yet (Anderson 2007), but as technology advances (brain scanning in particular) and becomes more sophisticated, it is becoming more and more feasible to do this mapping.

ACT-R has been fairly extensively tested against psychology data to validate its predictions. Like all information processing models, it has mostly been used in thought experiments and as a research tool. ACT-R in particular, though, has been used to create a large number of user models for several different domains including driving (Salvucci 2006), human-robot interaction (Ritter et al. 2006, 2007), aviation (Byrne and Kirlik 2005; Gluck et al. 2007), air traffic control and dual tasking (Schoelles and Gray 2001), and menu use (Byrne 2001). These models are harder to create than GOMS or KLM models, but they make more detailed predictions, including what information is required to perform the task, and the results of the information processing. If an addition is performed by users, for example, ACT-R can be used to predict their answers and the distribution of errors in their answers (Lebiere and Anderson 1998).

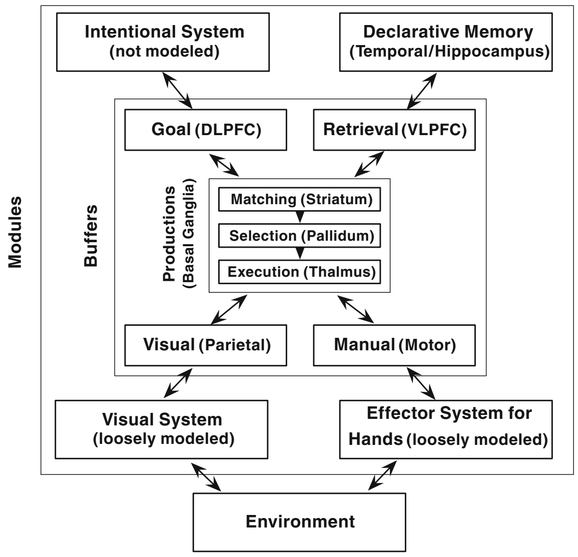

Fig. 14.3 The schematic block diagram of ACT-R 6. Taken from Anderson (2007) and used with permission

14.3.3 Summary

What is the purpose of user models? One use is that they can provide a way to create a shared representation of the user between interface designers, system designers, and others working on system development (Pew and Mavor 2007). They can also be used as summaries of knowledge (Newell 1990), and are thus useful to people learning about users. These models are also useful in video games and other simulations where the more advanced types can serve as surrogate users, opponents, and teammates (Pew and Mavor 1998; Ritter et al. 2003).

User models provide frameworks, ways of seeing designs and targeting potential problems, and integrating psychological principles because the user's knowledge and capabilities are represented explicitly.

Which type of user model you choose depends on what part of the design you are working on and what you want from the model. For example, task analysis as a simple version of a user model is very lightweight and very useful, particularly for simple web site design. A more complex and formal model is useful as the complexity and impact of the system increase. For testing large numbers of interfaces (e.g., a web site), or for testing a site often or repeatedly, an automatic tool is useful. For high impact interfaces (e.g., aviation, automotive interfaces due to the large number of users, or space station interfaces), more complete and complex models are useful because they are more accurate, but currently come at a very high price.

The choice of which model to use can be based on what you want to learn about how users interact with your system. This can be seen as a way to reduce system development risks, and your choice of model can be based on the questions of what risks do you want to reduce, will the system be fast enough to use? Fast enough to learn? Will the error rate be too high? Using risk reduction in system design is taken up in the next section.

14.4 Risk-Driven Incremental Commitment Model

14.4.1 Introduction

There is now widespread acceptance of the fact that most systems development follows an iterative cycle, often represented by the spiral model (Boehm and Hansen 2001). It is only relatively recently, however, that human factors issues have been explicitly incorporated into the latest version of the spiral model (Boehm and Lane 2006) by the (US) Committee on Human-System Design Support for Changing Technology (Pew and Mavor2007). The revised model—the Risk Driven Incremental Commitment Model (RD-ICM)—encourages incremental development of systems in an ongoing spiral process comprising requirements specification, technical exploration, and stakeholder commitment. The process is shown in Fig. 14.4, where movement around the spiral represents time and commitment and work on the project.

The spiral is also sometimes shown linearly, as in Fig. 14.4, for discussion purposes. At each stage, the system development is assessed for risks to the system's success. The process is then targeted at reducing these risks. Where the risks are technical (e.g., Can we build it? Can we build it for that price?), technical work is performed to reduce the risk through increased understanding of the technical issues and how to deal with them. Other risks can arise from historical events, which are harder to reduce; and from financial matters, which can often be reduced by setting up contracts at a known price.

The RD-ICM has several key features:

1. Systems should be developed through a process that considers and satisfices the needs of stakeholders (it finds a reasonable solution that keeps in mind the costs of finding a (better) solution). This step is addressed by the Exploration and Valuation stages shown in Fig. 14.4.
2. Development is incremental and performed iteratively. These related aspects are shown in Fig. 14.4 by the multiple loops representing the increasing amount of resources committed to design and implementation, and in Fig. 14.5 by the five stages (Exploration, Valuation, Architecting, Development, and Operation). These stages are incremental because movement from one stage to the next depends upon a successful review to go to the next stage.
3. Development occurs concurrently, that is, multiple steps may be performed simultaneously—some people thus refer to this model as an Incremental Concurrent Commitment model. One part of the system may be implemented while another part is being tested. This is not immediately clear from Figs. 14.4

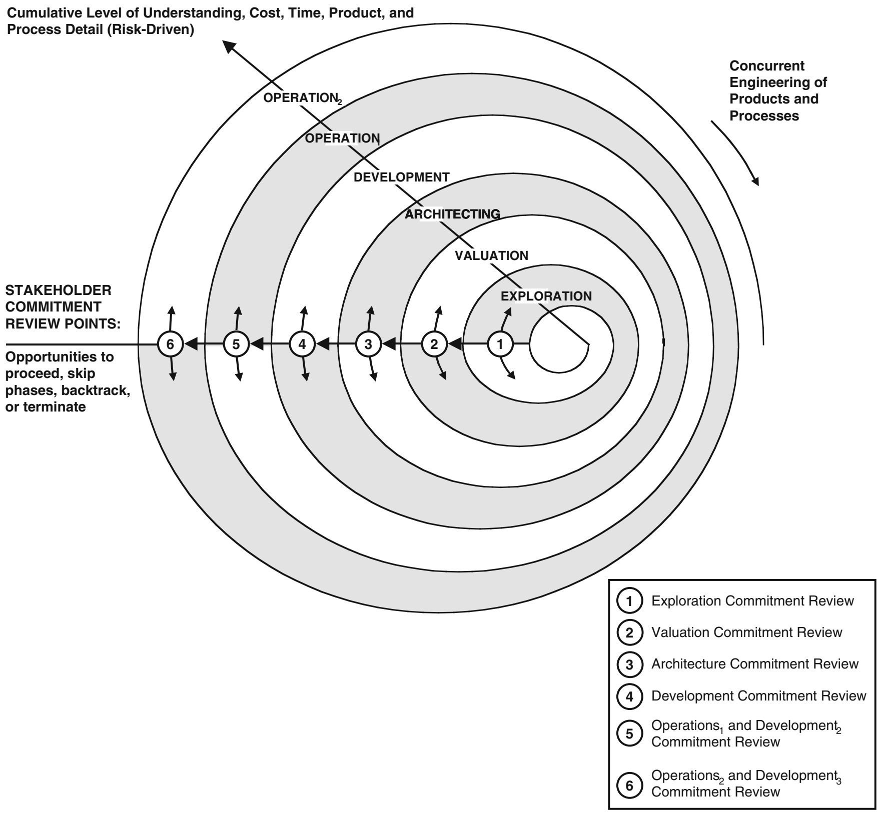
Cumulative Level of Understanding, Cost, Time, Product, and Process Detail (Risk-Driven)
Concurrent Engineering of Products and Processes
OPERATION2
OPERATION1
DEVELOPMENT
ARCHITECTING
VALUATION
EXPLORATION
STAKEHOLDER COMMITMENT REVIEW POINTS:
Opportunities to proceed, skip phases, backtrack, or terminate
6
5
4
3
2
1
1 Exploration Commitment Review
2 Valuation Commitment Review
3 Architecture Commitment Review
4 Development Commitment Review
5 Operations1 and Development2 Commitment Review
6 Operations2 and Development3 Commitment Review

Fig. 14.4 The RD-ICM model as a spiral. Reprinted from Pew and Mavor (2007) with permission from the National Academy of Sciences, courtesy of the National Academies Press

or 14.5, but can be seen more clearly in Fig. 14.6. This figure shows how the amount of effort put into a given activity varies through the life of a system for a hypothetical example system. Some peaks occur on some activities when their phase is active (e.g., development), and some activities peak near the reviews (i.e., the Evaluation, Valuation, Architecting, Construction, and Operations Commitment Reviews). The level of activity will also vary within a phase, as iterations are done within that phase.

4. The process explicitly takes account of risks during system development and deployment. The level of risk is assessed at reviews between stages (shown holistically in Fig. 14.4 and explicitly in Fig. 14.5). Risk is used to manage the project—the level of effort and level of detail of work are driven by the level of risk (Boehm 2008 provides some nice additional information and examples). Where there is no risk to system development, there is no need for any effort to reduce risk. For example, if the system being developed is similar to an existing product, there may be no reason to explore further how to support users or how to manufacture that system.

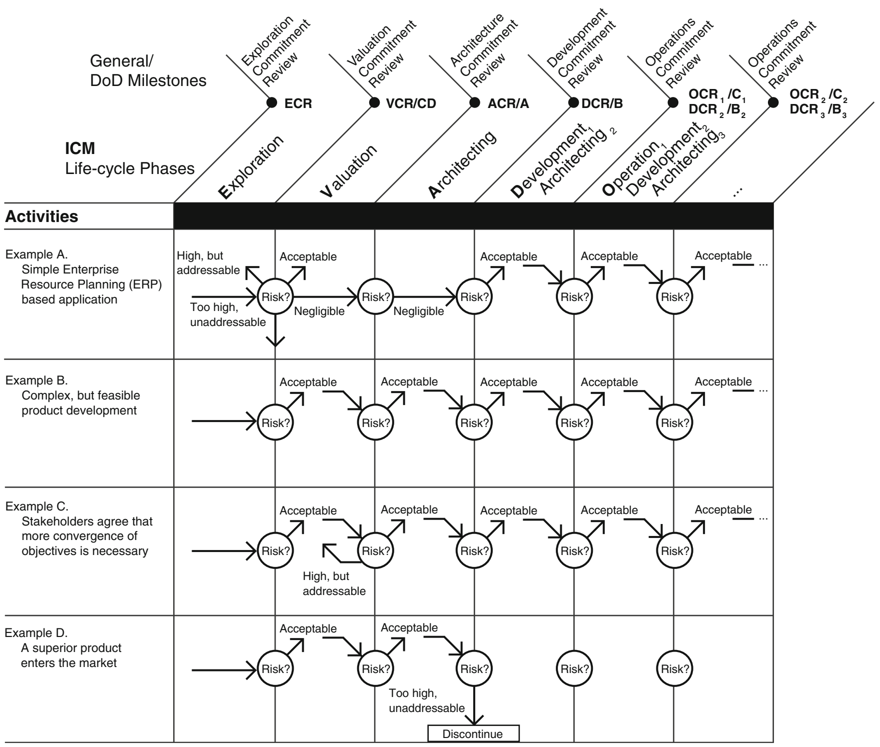
General/ DoD Milestones
Exploration Commitment Review
ECR
Valuation Commitment Review
VCR/CD
Architecture Commitment Review
ACR/A
Development Commitment Review
DCR/B
Operations Commitment Review
OCR1/C1 DCR2/B2
Operations Commitment Review
OCR2/C2 DCR3/B3
ICM Life-cycle Phases
Exploration
Valuation
Architecting
Development1 Architecting2
Operation1 Development2 Architecting3
...
Activities
Example A. Simple Enterprise Resource Planning (ERP) based application
Example B. Complex, but feasible product development
Example C. Stakeholders agree that more convergence of objectives is necessary
Example D. A superior product enters the market
Risk?
Acceptable
High, but addressable
Too high, unaddressable
Negligible
Discontinue

Fig. 14.5 The RD-ICM laid out linearly. This figure also shows the role of risk in system development, showing how different risk patterns yield different processes. Reprinted from Pew and Mavor (2007) with permission from the National Academy of Sciences, courtesy of the National Academies Press

Pew and Mavor (2007, pp. 91–133) provide three examples of using the Risk-Driven Spiral Model method to develop specific systems. These examples are taken from different domains and are different sizes. Their Chap. 5 (http://www.nap.edu/openbook.php?record_id=11893&amp;page=91) covers the development of an unmanned aerial system (i.e., an unmanned aerial vehicle, a UAV) for the military, a port security screening system for homeland security, and an intravenous infusion pump for use in hospitals. Each example covers different aspects of the process, so together they provide fairly broad coverage of the approach.

The model is relatively complex, so we advocate that another way to understand it is through the viewpoint of concurrent activities. The standard way of presenting the model is as a spiral (as shown in Fig. 14.4), although the spiral can be unwound and the task activity levels can be represented in a linear fashion (Fig. 14.5), which makes it easier to recognize the concurrent phases of development, as well as concurrent activities that are out of phase and aspects like stakeholder commitment review points where the risks are assessed.

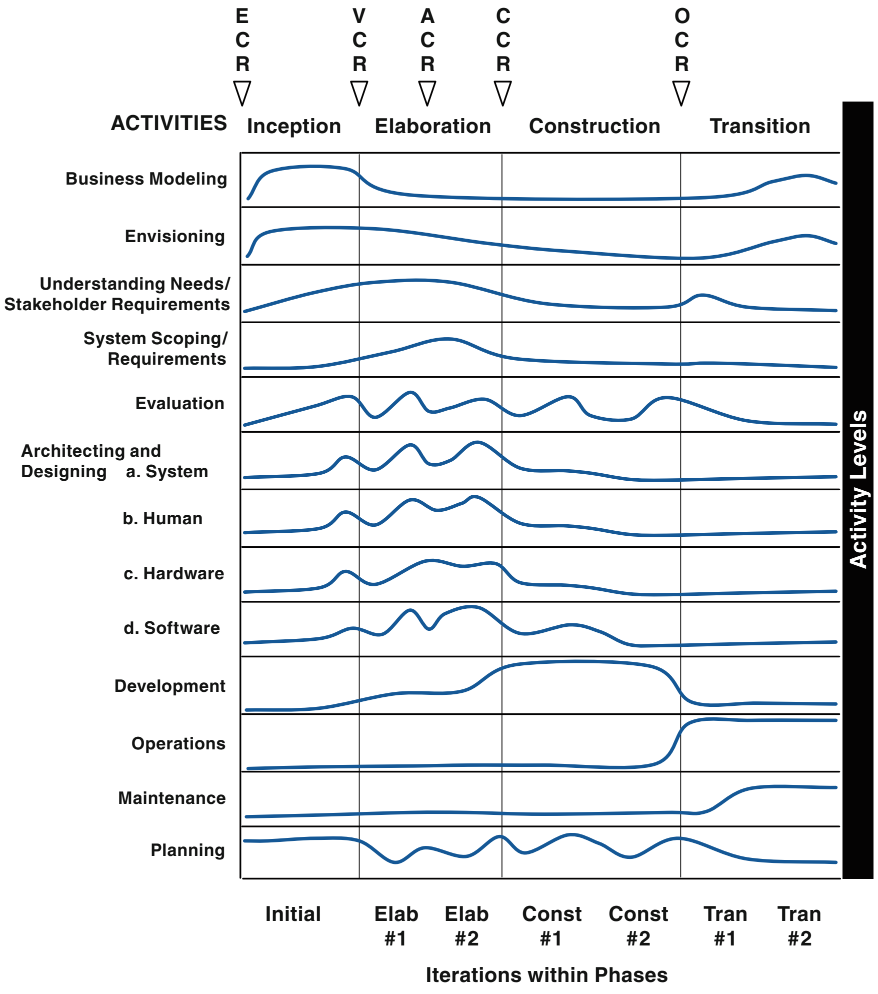

Fig. 14.6 The role of concurrency in system development, showing how different tasks may be performed concurrently and how activity levels rise and fall over the course of a project. It is similar to Fig. 2.3 in Pew and Mavor (2007) and incorporates ideas from Boehm's other work (e.g., Boehm 2008)

14.4.2 Insight 1: The RD-ICM Provides a Way to Organize User-Related Knowledge and Ways of Knowing

Experiences teaching and applying the RD-ICM to system design have led to a few insights and extensions. These are related to learning: learning through using this approach how to organize methods to reduce risks, learning by system development managers that there are sometimes risks related to humans using their systems, learning that designers are stakeholders too, and learning by designers as lessons from one design are applied to later designs.

Pew and Mavor (2007) report that the RD-ICM approach provides a useful way to organize usability methods. We have found that this approach also provides a useful framework for teaching this material. The three main areas that involve HIS activities are identified as:

1. Defining the opportunities and context of system use.
2. Defining system requirements and design.
3. Evaluation of the system.

There are several methods that can be used to reduce risk in these three areas. All of the HCI methodologies (not just the examples presented in the Pew and Mavor book) can be organized with respect to how much they can contribute to each stage of system development.

The RD-ICM has been very useful in discussing the relative merits of methodologies, and when to use each of the methodologies or techniques. Figure 14.7 highlights several examples of methods that are applicable across several stages of system development, as shown by the horizontal lines under each of the method names. The figure could be further extended by weighting the lines to emphasize where individual methods are more appropriate to a particular stage of development. If all the methods were included, the figure would also show that methods (and thus practitioners) could be grouped by usefulness at particular stages of development: some methods are best suited to the early valuation stage, for example, and some to evaluation.

14.4.3 Insight 2: RD-ICM is Descriptive as Well as Prescriptive

The RD-ICM formalizes to some extent what many people accept as perceived wisdom, i.e., that many system developers already recognize that several development processes are risk driven (or at least risk aware), incremental, and concurrent. Indeed, we believe that most system development processes are risk driven, and that systems developers are aware of the risks and adjust their development processes to reduce or mitigate the effects of the identified risks. We also believe that some parts of system development are often carried out in parallel. Furthermore, we believe that there is buy-in from at least some of the system stakeholders in nearly all projects. The RD-ICM is therefore not merely a normative model, prescribing what system developers should do, but instead captures and describes the practice of systems development. If the RD-ICM was described to systems developers, we believe many of them would claim that they already follow a similar process.

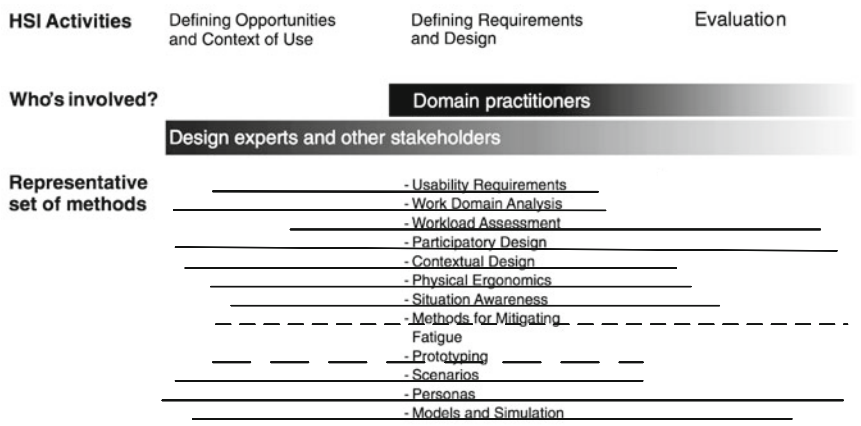

Fig. 14.7 Figure 7.1 from Pew and Mavor (2007) revised to show the approximate range of use of several methodologies across the development process. Reprinted with permission from the National Academy of Sciences, courtesy of the National Academies Press

There are two major areas of risk, however, that system developers and managers seem to be less aware of:

- The risks that arise from not giving appropriate consideration to *all* of the system stakeholders
- The risks that arise from not considering humans as part of the system.

Developers and managers are probably not effectively addressing these risks because they believe the risks have low probability or only lead to very minor consequences, perhaps because they lack formal training in these areas.[1] We believe that the developers and managers more fundamentally do not recognize the potential threat to the system's success from these two major areas of risk. Thus, we can either educate the managers, or provide other ways to highlight risks outside their expertise. It may be useful to bring in outside experts to evaluate the risks. The corollary of this is that it is probably worthwhile to include outside experts in the evaluation of all risks. They are likely to have greater awareness of a wider range of risks across a wider range of contexts, and to be more objective with respect to the project, so they will help to make the evaluation more comprehensive.

Irrespective of whether you are following some version of the Spiral Model (such as the ICM or RD-ICM) or any other life cycle model (waterfall, V-model, and so on), it is important that all stakeholders are considered. It is also important

[1] Engineers will see engineering risks, accountants accounting risks, and human factors engineers HF risks.

that they are considered from the start of the project. In other words, before we get to thinking about design issues.

There are several illustrations of why it is important to include consideration of users at the earliest stages of system development. Figures 1.1 through 1.4 in Chap. 1, for example, all showed that there are fundamental aspects of human behavior that designers are often unaware of. Some would argue that the consequences of these failures are irritating rather than severe, but the range and impact of the risks are exemplified in summary reviews such as those by Booher and Minninger (2003) and Casey (1998, 2006), in the regularly updated Risks Digest,[2] and in publicly reported incidents like that involving the USS Vincennes, where a civilian airliner was shot down by a US Navy ship because the airliner was thought to be a fighter plane. The examples in all of these reviews help to raise the visibility of the risks that need to be considered during development.

This approach is therefore not just prescriptive but also descriptive. Failures arise from not knowing risks. To fix this, we need to educate designers about human-centered risks and how to avoid them. This book can be seen as a direct step to address this problem.

14.4.4 Extension 1: Designers are Stakeholders Too

One of the insights that arose from teaching the RD-ICM is that a description of the development process reinforces the fact that designers are stakeholders too (see Steele et al. 2011 for one example). The success of designers is at least partially linked to the success of the project, and their needs and capabilities will influence the development process.

Designers are therefore stakeholders too, and, just like users, they have capabilities and limitations that can affect the outcome of a project. They have creative skills and experience of how to turn requirements into deliverable systems. More fundamentally, they will also have experience of the feasibility of whether a particular set of requirements can reasonably be turned into a deliverable system. In one sense, designers are users too, because they often rely on automated tools to help them to carry out and manage system development. If they are not supported or lack resources, including time, they will produce problems like that shown in Fig. 14.8.

If designers are stakeholders then there needs to be time allocated to soliciting their input, and to making sure that they are equipped with the appropriate skills and tools to do the design. This may mean that they need to be educated about the capabilities and limitations of users, for example, and to be provided with tools that can allow them to create design representations that they can share with other stakeholders as well as the users.

[2] http://catless.ncl.ac.uk/Risks

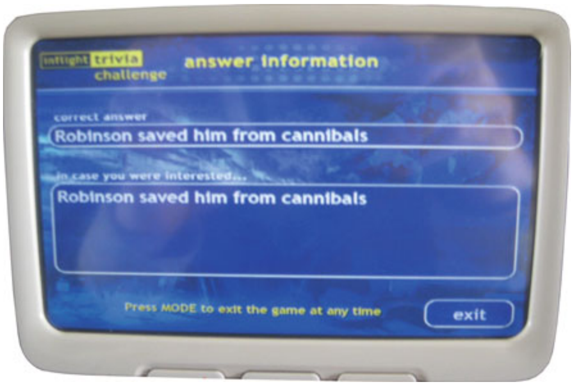
inflight trivia challenge
answer information
correct answer
Robinson saved him from cannibals
in case you were interested...
Robinson saved him from cannibals
Press MODE to exit the game at any time
exit

Fig. 14.8 The question was, "How did Robinson Crusoe meet Friday?" The duplication of the answer into the extra information field suggests that the interface did not support a system designer, and that the results were not tested

When designers (and implementers) push back against the discussion and management of risks related to human–system integration, this may be because they do not fully appreciate the risks in this area. Explicitly including designers as stakeholders should focus on helping them become more cognizant of the issues and context involved that is outside their training. Giving them this broader context should ultimately lead to better integration and greater risk reduction, including the reduction of system implementation related risks.

14.4.5 Extension 2: Learning Within and Between Projects

Given the multidisciplinary nature of system development, and the importance of involving all the stakeholders, it is obvious that communication is important. The various disciplines that need to talk to each other during system development all have their own history and culture, which includes their own terminology. One of the easiest ways of facilitating communication is through the use of representations that help to make ideas and concepts more tangible. Pew and Mavor (2007) explicitly recognized this in their call for research on shared representation and integration across the design process. In some ways this can be regarded as an extension to the idea of software design patterns which provide generic templates or descriptions of solutions to problems that can be applied in different situations (e.g., Gemma et al. 1995).

One of the major results from using simulation in CAD/CAM is that the lessons learned about one design can be applied to subsequent designs as well as to the current design. For example, in printed circuit boards, traces (printed wire paths) can be positioned too closely together. Tools like Spice (Thorpe 1992) help to highlight effects such as cross talk between the traces and short circuits that can be caused by the manufacturing process. After these design lessons are learned on the first design, they can be used to steer subsequent designs away from the problem rather than forcing the designer to start from scratch each time, modifying their designs based on a new analysis of the problems. Similar effects of transfer will happen after computing Fitts' Law a few times or comparing mouse movement times to keystroke times.

To facilitate learning, shared representations need to be made available within projects, across the system development team and across the different stages of the design process for a single project. We extend this, though, to re-using representations across multiple projects. This may mean that the shareable representations have to be able to reflect different levels of abstraction (based on their dependency on their context of use) so that they can be learned from and re-used. The corollary of this is that those people developing the shared representations are likely to need to document the shared representations with information about context.

In addition to sharing representations within projects, it may be possible to re-use these representations across projects. This would increase the value of these representations, and achieve the re-use that is called for by Pew and Mavor (2007). This would also broaden the audience for the shared representations to include managers and developers on future projects who may be able to learn about potential risks and which stakeholders to consult.

14.4.6 Summary

The RD-ICM model is just one way of creating systems that are useful and safe. It involves identifying the stakeholders—including the designers—and coming up with a system through satisficing their different requirements, finding the best solution matching their constraints, including the cost to find better solutions as a constraint. If stakeholders' needs are not adequately supported, they may not participate in the system development process fully, and may either obstruct the process or even refuse to accept the final system. An advantageous side effect of using the RD-ICM is that it can provide new ways of summarizing HCI methods and, more broadly, Human-System Integration (HSI) as an integral part of design and development.

In the vast majority of systems, a failure to consider the users as stakeholders, and a lack of understanding of the abilities, capabilities, and limitations of users will constitute a risk to system development and use. The main exceptions are where the system being developed is not novel, so the risks in these areas are likely to have been considered and dealt with on other projects. Where there are risks, however, work has to be done to manage them appropriately, given the available resources, and to balance all of the identified risks (user related risks will need to be balanced against technical risks, for example).

Managers and developers are usually familiar with the technical risks associated with software and hardware. To make them more aware of user-related risks, they are likely to require some level of education and training about the capabilities and limitations of users. This education and training should take them to a level where they feel comfortable discussing these sorts of issues with people from the other disciplines involved in systems development.

Shared representations are an important part of the development process, but they can be expensive to produce. It is therefore important to find ways to make them more valuable through reuse, for example. One way of doing this is by applying lessons from one design to subsequent designs on related projects.

On the surface, the RD-ICM may appear to encapsulate a relatively complex theory of how systems should be created. It attempts to capture best practice and what many developers already do, highlighting that development is not a simple, linear, one-size-fits-all process. There are enough facets to the RD-ICM that it appears to have the same sort of level of complexity as a programming language or even a cognitive architecture. Although learningto use the RD-ICM approach takes time and effort, the fact that it takes explicit account of human factors currently makes it possibly the best model to use for development.

14.5 Building on the Foundations

At this point you should know much more about users than you did when you started reading this book. We hope that we have convinced you of the importance of understanding users and that, as a result, you are now more sensitive to the ways that users think and behave.

There *is* a lot to learn, and while we have presented you with a lot of information, we continue to learn more about users as new technologies emerge and give rise to new ways of working. Even with the new technologies and new ways of working, it is important to think of them in terms of *particular* users doing *particular* tasks in a *particular* context. The information we have presented in this book should allow you to start to do that in a principled way, enabling you to design usable systems *and* to justify why you have designed them in a particular way. The *Implications for system design* sections should help here. Note, however, that in this book we have only really scratched the surface of what there is to learn about people, tasks, and contexts. If you want to find out more about any of the topics we mention, the lists of *Other resources* should provide a good starting point. Cynics might say that "Keep in mind that a year in the lab can save you an hour's reading. That is, spending a little time looking at previous work can save a lot of time needlessly duplicating known results.

One implicit lesson that we hope you have learned is that developing systems draws on multiple disciplines. We are not expecting software designers to become fully fledged psychologists, but we hope that our book sensitizes software designers to the psychological issues (about users) that need to be considered. We also hope that it provides a foundation for useful dialog across disciplines during system design.

You should also have tools for finding out information when the information is not yet published. The short exercises at the end of most chapters on gathering data and Chap. 13 on usability studies should provide you with a way to find out more about particular users, particular tasks, or particular aspects of a task. There are important topics not included here because of space, such as emotions, and these you are now well equipped to learn on your own, or through further formal study

As you start to use and apply your knowledge of users to systems development, you are likely to find that it leads you to raise several questions. When you address these questions to human factors engineers, you will often find that their answer is "It depends." If this book has achieved nothing else, it should at least have helped you to appreciate *why* "it depends": system performance is all about particular people doing particular tasks in particular contexts, and those people are all different individuals, they have different skills and abilities, and they may work in different physical, social, and organizational contexts.

The importance of appropriately integrating what we know about users into system design is becoming increasingly widespread. The revised version of Boehm and Hansen's (2001) incremental commitment model of system development, for example, includes explicit consideration of users (Pew and Mavor2007). The new model provides a principled way for deciding when you need to know more about your users. It explicitly acknowledges that not knowing enough about your users is a risk to the successful completion of a system development project. If this lack of knowledge about your users (their tasks and the context in which they work) poses a significant risk, then you need to invest more time and effort in addressing those issues until the risk is reduced to an acceptable level.

We have now shown you what sort of things you will need to know about your users, and how to go about finding out that information. In doing so, we have provided you with the foundations for designing user-centered systems.

14.6 Other Resources

Pew (2007) has written a history of models in this area. In it he provides a summary as well as a description of the relationships between the various families of models.

Salvucci has some nice models of how people drive (Salvucci2006), and some tools for predicting how secondary tasks will impair driving (Salvucci et al. 2005). Byrne and Kirlik (2005) have provided similar lessons in aviation.

We have previously laid out a vision of future work that is needed to help designers create models routinely (Ritter et al. 2003). It is available online and notes about 20 projects that remain to be done with computational models of users.

Pew and Mavor's (2007) book is available on line, and you can purchase it or register and download it. Their book provides useful pointers to a range of methodologies for reducing the risks to system success that are caused by not understanding users or their tasks. It could be and has been used to teach a separate class on HCI methodologies. There was also a special issue of the *Journal of Cognitive Engineeringand Decision Making* related to the topic published in 2008.

A short, easy to read technical report is available online that lays out a taxonomy of system development risks and a method to perform a risk analysis:

Carr, M., Konda, S., Monarch, I., Ulrich, C., & Walker, C. (1993). Taxonomy-Based Risk Identification (Tech. Report No. CMU/SEI-93-TR-6, ADA266992). Pittsburgh, PA: Software Engineering Institute, Carnegie-Mellon University.

14.7 Exercises

14.1 Consider a smartphone, either a specific one or a composite one, or a simple cell phone. Note all the limitations that will preclude users from performing well with it. Organize these by the ABCS framework.

14.2 Choose an existing interface. Note the tasks that users will want to use it for. Note the types of users and their characteristics. Run two small studies examining how well your theory of use fits the data. These studies could be to find out what tasks users really do, how well they do it (time and errors, and strategies), or characteristics of the users.

14.3 Choose an interface or system to create, such as a new application for a smartphone, such as a game, or book reader. Note what risks there are in developing such a system, with particular attention paid to the technical risks and those related to the user. Prioritize them. If you had 100 h, how would you allocate that time to reduce these risks?

14.4 Go back and read the Appendix on the Kegworth air accident again. Read the supplementary material referenced (e.g., the AAIB report) or other reports you can find. Identify four more factors (events, processes, or mistakes) that can be considered as causes of the accident. Describe how these things could have been avoided or ameliorated. Address the question of whether it was 'pilot error' that caused that plane to crash.

References

Anderson, J. R. (2007). *How can the human mind exist in the physical universe?* New York, NY: Oxford University Press.

Barnard P. J. (1987). Cognitive resources and the learning of human-computer dialogues. In J. M. Carroll (Ed.), *Interfacing thought: Cognitive aspects of human–computer interaction* (pp. 112–158). Cambridge, MA: MIT Press.

Baxter, G., Besnard, D., & Riley, D. (2007). Cognitive mismatches in the cockpit: Will they ever be a thing of the past? *Applied Ergonomics, 38*, 417–423.

Boehm, B. (2008). Making a difference in the software century. *IEEE Computer,41*(3), 32–38.

Boehm, B., & Hansen, W. (2001). The spiral model as a tool for evolutionary acquisition. *Crosstalk: The Journal of Defense Software Engineering, 14*(5), 4–11.

Boehm, B., & Lane, J. (2006). 21st century processes for acquiring 21st century systems of systems. *Crosstalk, 19*(5), 4–9.

Booher, H. R., & Minninger, J. (2003). Human systems integration in Army systems acquisition. In H. R. Booher (Ed.), *Handbook of human systems integration* (pp. 663–698). Hoboken, NJ: John Wiley.

Byrne, M. D. (2001). ACT-R/PM and menu selection: Applying a cognitive architecture to HCI. *International Journal of Human–Computer Studies, 55*(1), 41–84.

Byrne, M. D., & Kirlik, A. (2005). Using computational cognitive modeling to diagnose possible sources of aviation error. *International Journal of Aviation Psychology, 15*(2), 135–155.

Card, S. K., Moran, T. P., & Newell, A. (1980). The keystroke-level model for user performance time with interactive systems. *Communications of the ACM, 23*(7), 396–410.

Card, S. K., Moran, T., & Newell, A. (1983). *The psychology of human–computer interaction*. Hillsdale, NJ: Erlbaum.

Casey, S. M. (1998). *Set phasers on stun: And other true tales of design, technology, and human error*. Santa Barbara, CA: Aegean.

Casey, S. M. (2006). *The atomic chef: And other true tales of design, technology, and human error*. Santa Barbara, CA: Aegean.

Ericsson, K. A., & Simon, H. A. (1993). *Protocol analysis: Verbal reports as data* (2nd ed.). Cambridge, MA: MIT Press.

Gemma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Boston, MA: Addison-Wesley.

Gluck, K. A., Ball, J. T., & Krusmark, M. A. (2007). Cognitive control in a computational model of the predator pilot. In W. D. Gray (Ed.), *Integrated models of cognitive systems* (pp. 13–28). New York: Oxford University Press.

Ivory, M. Y., & Hearst, M. A. (2001). The state of the art in automating usability evaluation of user interfaces. *ACM Computing Surveys, 33*(4), 470–516.

Kieras, D. E. (1999). *A guide to GOMS model usability evaluation using GOMSL and GLEAN3*. AI Lab, University of Michigan. Retrieved 10 March 2014 from http://web.eecs.umich.edu/~kieras/docs/GOMS

Kieras, D. E., Wood, S. D., Abotel, K., & Hornof, A. (1995). GLEAN: A computer-based tool for rapid GOMS model usability evaluation of user interface designs. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'95)* (pp. 91–100). New York, NY: ACM.

Kieras, D. E., Wood, S. D., & Meyer, D. E. (1997). Predictive engineering models based on the EPIC architecture for a multimodal high-performance human-computer interaction task. *Transactions on Computer–Human Interaction, 4*(3), 230–275.

Laird, J. E. (2012). *The soar cognitive architecture*. Cambridge, MA: MIT Press.

Landauer, T. K. (1987). Relations between cognitive psychology and computer systems design. In J. Preece & L. Keller (Eds.), *Human–computer interaction* (pp. 141–159). Englewood Cliffs, NJ: Prentice-Hall.

Lebiere, C., & Anderson, J. R. (1998). Cognitive arithmetic. In J. R. Anderson & C. Lebière (Eds.), *The atomic components of thought* (pp. 297–342). Mahwah, NJ: Erlbaum.

Morrison, J. E. (2003). *A review of computer-based human behavior representations and their relation to military simulations* (IDA Paper P-3845). Alexandria, VA: Institute for Defense Analyses.

Newell, A. (1990). *Unified theories of cognition*. Cambridge, MA: Harvard University Press.

Pew, R. W. (2007). Some history of human performance modeling. In W. Gray (Ed.), *Integrated models of cognitive systems* (pp. 29–44). New York, NY: Oxford University Press.

Pew, R. W., & Mavor, A. S. (Eds.). (1998). *Modeling human and organizational behavior: Application to military simulations*. Washington, DC: National Academies Press. Retrieved from 10 March 2014 http://books.nap.edu/catalog/6173.html

Pew, R. W., & Mavor, A. S. (Eds.). (2007). *Human-system integration in the system development process: A new look*. Washington, DC: National Academies Press. Retrieved March, 2014 from http://books.nap.edu/catalog.php?record_id=11893

Ritter, F. E. (2003). Soar. In L. Nadel (Ed.), *Encyclopedia of cognitive science* (Vol. 4, pp. 60–65). London: Nature Publishing Group.

Ritter, F. E., & Bibby, P. A. (2008). Modeling how, when, and what learning happens in a diagrammatic reasoning task. *Cognitive Science, 32*, 862–892.

Ritter, F. E., Shadbolt, N. R., Elliman, D., Young, R. M., Gobet, F., & Baxter, G. D. (2003). *Techniques for modeling human performance in synthetic environments: A supplementary review*. Wright-Patterson Air Force Base, OH: Human Systems Information Analysis Center (HSIAC).

Ritter, F. E., Van Rooy, D., St. Amant, R., & Simpson, K. (2006). Providing user models direct access to interfaces: An exploratory study of a simple interface with implications for HRI and HCI. *IEEE Transactions on System, Man, and Cybernetics, Part A: Systems and Humans, 36*(3), 592–601.

Ritter, F. E., Kukreja, U., & St. Amant, R. (2007). Including a model of visual processing with a cognitive architecture to model a simple teleoperation task. *Journal of Cognitive Engineering and Decision Making, 1*(2), 121–147.

Salvucci, D. D. (2006). Modeling driver behavior in a cognitive architecture. *Human Factors,48*, 362–380.

Salvucci, D. D., Zuber, M., Beregovaia, E., & Markley, D. (2005). Distract-R: Rapid prototyping and evaluation of in-vehicle interfaces. In *Human Factors in Computing Systems: CHI 2005 Conference Proceedings* (pp. 581–589). New York, NY: ACM Press.

Schoelles, M. J., & Gray, W. D. (2001). Argus: A suite of tools for research in complex cognition. *Behavior Research Methods, Instruments, & Computers, 33*(2), 130–140.

Steele, M., Dow, L., & Baxter, G. (2011). Promoting public awareness of the links between lifestyle and cancer: A controlled study of the usability of health information leaflets. *International Journal of Medical Informatics, 80*, e214–e229.

Thorpe, T. W. (1992). *Computerized circuit analysis with SPICE: A complete guide to SPICE, with applications*. New York, NY: Wiley.