# Gombe State University
# Faculty of Science
# Department of Mathematics
# COSC 409: Human Computer Interaction (HCI)

**Course lecturer:** Dr. Haruna Chiroma

**Course outline**

- Foundation of HCI, principles of GUI, GUI toolkits
- Human centred software development and evaluation
- GUI design and programming
- Design principle for interactive systems
- Web usability and accessibility

## Introduction

The last part of the twentieth century witnessed an unprecedented growth in computing and information. The resulting explosion of computing innovation has reshaped the modern world, entering homes, businesses, entertainment, and communication. This growth has made computers more accessible and useful. As computing attempts to serve more people in more ways than ever before, the human-computer interface is of critical importance. This note is about the technologies that shape the human-computer interface. The goal is to present not only principles for implementing today's interactive systems, but also to prepare for future models of interaction.

Word processors, spreadsheets, e-mail, and Web browsers are all interactive applications that keep us in touch with each other and provide us with information far beyond what we can handle on our own. Computing can simplify and enrich our lives in so many ways. Cell phones, personal digital assistants (PDAs), music players, and laptops have moved into our way of life. Text messaging has become a serious form of social interaction among teens. All of these require the implementation of interaction systems. These are only the applications that we know. The future holds much more. To get a grasp on the future of interactive computing, we must look at the forces that drive computing in general. The growth in computing is best characterized by Moore's Law 1, which says that every 18 months computers will be twice as fast and have twice as much memory with no increase in cost. Gordon Moore first stated his law in 1965 and it has been true since then. It is widely believed that although we will eventually reach the barriers of our known physics, this law can continue to hold for two more decades.

This explosive growth, with no increase in cost, continues to radically change the shape and nature of computing. In the late 1960s and early 1970s, computing was characterized by huge machines occupying large, specially air-conditioned rooms and served by specially trained personnel. User interaction was primarily via punched cards with cumbersome, expensive terminal devices being introduced later. In the 1980s, computers began to dominate the desktop and at the turn of the century, a person could carry more computing in a shirt pocket than a 1000-square-foot room once held. A simple way to think of Moore's Law is that computing capacity will increase roughly 10 times in five years and 100 times in 10 years. Thus, the average laptop of the year 2010 will execute approximately 20 billion instructions per second using 20 GB of RAM. Moore's Law is not the only exponential growth in computing. Growth in disk space has exceeded that of computing power. The number of computers on the Internet has grown even faster. All of these developments bring the power to create, manipulate, and communicate information into the hands of ordinary people. It is a central thesis of this note that computing exists to serve the needs of human beings. The exponential growth in computing stands in stark contrast to the growth in human capacity to perceive, remember, and express information. The curve describing the change in human capacity over time is flat. Our personal capabilities are not changing. At first, this might be a bit discouraging until you consider how great human capacity really is. The human brain contains several orders of magnitude more memory and computing capacity than the largest computers ever built.

When it comes to sophisticated information processing, human beings dominate. So what, then, what are the advantages of computers over people? Computers are more precise. A computer's memory is exact rather than approximate. Computation produces a crisp rather than a fuzzy result. We might use "fuzzy" reasoning in our programs, but the result is always a particular fuzzy value. Computers and their networks assist us in transcending the barriers of time and space. They can perform tasks without us paying attention. Computers can perform the repetitive and boring work without getting tired. In many ways, computers complement rather than replace the capabilities of human beings.

The things computers do for us are frequently those things that as humans we enjoy the least or perform poorly. The human-computing team forms a powerful combination in seeking solutions to problems. To understand the future of interactive systems, a more interesting curve is the inverse of Moore's Law. If, instead of holding cost constant, we hold computing capacity constant and look at the change of price, the cost of a unit of computing is cut in half every 18 months or is 100 times cheaper in 10 years.

It is this inverse law that brought us the personal computer and the PDA. In the year 2008, a computer that can perform speech recognition cost $700. In the year 2018, speech recognition will be a $7 part that can be included in virtually any device. The inverse of Moore's Law is very important when we consider the future of interactive systems. For both human and economic reasons, the dominant model for interacting with a computer is seated at a desk with a screen, keyboard, and mouse. For the recent past, the desktop computer is the most economical compromise between the capabilities of computers and the physical/perceptual capabilities of human beings. When powerful computing costs $7, the economic balance between humans and computing will shift just as powerfully as when the room-size computers arrived on the desktop. We can already see this trend toward embedding computing into the way people live, work, and play. As of this writing, the number of cell phones (which are computers) far exceeds the number of personal computers.

Cell phones are so inexpensive that they are given away as promotions to attract phone system customers. The number of smart phones or PDAs being sold has now exceeded the number of laptops sold. Computers will be everywhere. Interactive devices will be everywhere. At the end of the twentieth century, we witnessed a transformation from each computer served by hundreds of people to each person served by their own computer. This transition will occur again with each person served by hundreds of computers. This will radically change the way in which people interact with technology. In particular, interactive computers must blend into the physical situations in which people live and work. Interacting in a car is very different from interacting on the trading floor of a stock exchange, which is different from the needs of a mechanic repairing a computer-controlled engine, which contrasts with a search-and-rescue worker on an earthquake site. The computer must adapt to people rather than the other way around (Olsen, 2009).

## Foundations of HCI

From 'foundations' to 'new paradigms' is a wide canvas and this paper attempts to paint a picture of human–computer interaction from its earliest roots to future challenges. It is also iconic in that HCI as an academic discipline has always been positioned, sometimes uneasily, sometimes creatively, in the tension between solid intellectual rigour and the excitement in new technology. Stefano Levialdi, in who this special issue is in honour, had a rich appreciation of both and so I hope this paper is one that he would have enjoyed as well as offering an overview of the field as it was, as it is and as it could be.

Human–computer interaction, like any vocational discipline, sits upon three broad foundations.

Principles – First, and most obviously are the intellectual theories, models and empirical investigations that underlie the field. Give HCI's cross-disciplinary nature, some of these come form a number of related disciplines and some as core HCI knowledge.

Practice – Second, HCI is a field that, inter alia, seeks to offer practical guidance to practitioners in interaction design, usability, UX, or whatever becomes the next key term. However, also it is a discipline that has always sought to learn form the practical design and innovations that surround it.

People – Finally, there are the visionaries who inspire the field and perhaps most importantly the HCI community itself: researchers, educators and practitioners.

I will not attempt to separate these three in the following sections as they are all deeply intertwined in both the history and current state of HCI.

The interplay between the first two is central to the long-standing discussion of the nature of HCI originally posed by Long and Dowell [LD89]: is it a science, engineering or craft discipline? However, when I addressed the scientific credentials of HCI in my own response to this work [Dx10] in the IwC Festschrift for John Long, I found myself addressing as much the nature and dynamics of the academic community as the literature itself.

I will not reprise the arguments here, but the importance of the community is a message that is also central to Stefano's legacy. As well as deeply humane person at a one-to-one level, his contribution to the development of the Italian HCI community, and the founding of the AVI conference series have been of importance to many individuals as well as the academic growth of the field. It is not that the archival written outputs are not critical, indeed Stefano's role in JVLC is evidence of that, but that scientific outputs are always the result of a human process.

HCI developed as a discipline and a community in the early 1980s, triggered largely by the PC revolution and the mass use of office computers. It was in the early 1980s when the major HCI conferences began Interact, CHI, British HCI and Vienna HCI; all but the last still active today.

Core concepts were also formulated in those days including the notion of direct manipulation and user centred design [Sc83,ND86].

However, while the identifiable discipline began in the 1980s, the intellectual roots can be traced back at least 25 years earlier.

The graphical user interface and desktop metaphor, embodied in the early Apple Mac, were the result of work at Xerox PARC throughout the 1970s, mostly based around graphical programming environments such as Smalltalk and InterLisp, and leading to the design of the Xerox Star office computer [SK82, JR89], a conceptual breakthrough albeit a commercial failure.

Going back further, Sutherland's Sketchpad [Su63], the first graphical user interface dates back to the early 1960s and in the same period, Englebert's 'research centre for augmenting human intellect' was responsible for the invention of the mouse, as well as early versions of electronic conferencing [EE68].

However, the very first true HCI paper dates back into the late 1950s, with Shackel's 'Ergonomics for a Computer' [Sh59]. While Sutherland and Englebert were early examples of the vision/innovation side of HCI, Shackel's first HCI paper came more from a practical design perspective, the redesign of the control panel of EMIAC II, an early analogue computer.

Although the computer was analogue not digital, and the controls knobs and patch-panels, not mice or keyboards, many of the principles of practical usability engineering can be seen in this very earliest HCI paper including prototyping, empirical testing, visual grouping, and simplifying design. Furthermore this very practical work rooted itself in earlier theoretical work in ergonomics and applied experimental psychology, in many ways prefiguring the discipline we know today.

**Theory and Contributing Disciplines**

As already noted, the theoretical and empirical foundations of HCI draw partly from a number of related disciplines and partly are special to HCI itself.

In the earliest days the main disciplines involved in HCI were computer science, psychology and ergonomics, as reflected in Shackel's early paper. However, these disciplines were soon joined by social science, or, to be more precise, the ethnographic and anthropological side of sociology.

The input from ergonomics was initially in terms of physical ergonomics, sitting at a computer terminal, pressing keys; however, this more physical side of HCI declined rapidly as computers became commoditised as opposed to being in special settings and issues of physical ergonomics were relegated to health and safety concerns. To some extent this followed from the natural development of the area, once computers were mass-produced, practitioners had little control of the physical system unless they worked for major manufacturers. However, users have suffered from this loss of ergonomic input: many laptops and other devices sacrificed physical ergonomics for surface aesthetics, as a generation of RSI sufferers will attest! Happily, in more recent years, issues of physical design have resurfaced with interest in tangible computing and strong research connections developing with product design.

Another disciplinary connection that waned but is resurfacing is that of broader socio-technical design. Certain practitioners and researchers drew in the work of information scientists such as Enid Mumford [Ri06] and Checkland's softsystems methodology [CS99]. While this has had its legacy in HCI theory and practice, not least the focus on multiple stakeholders, the fields of information science and HCI have been largely parallel rather than interconnected except in the Nordic HCI tradition.

However, as with the physical side of HCI, there are signs of resurfacing interest in more organisational or community focused views of human activity. This is perhaps most marked in the related web science community where methods such as social network analysis are clearly very important, enabled by the vast quantities of data obtained form web-based systems. However, these techniques are also being used within HCI, not least Liu et al.'s study of HCI itself [LG14], which we'll return to later.

A few years ago Clare Hooper and I looked at the relationship between HCI and web science [HD12, HD13]. Although there are core differences in scope and focus, there are strong overlaps between the two. We drew on the web science 'butterfly', which includes all the disciplines that web science draws on. This was remarkably similar to those that connect with HCI differing mostly in the 'heat map' of those most active or relevant (see figs 1 and 2).

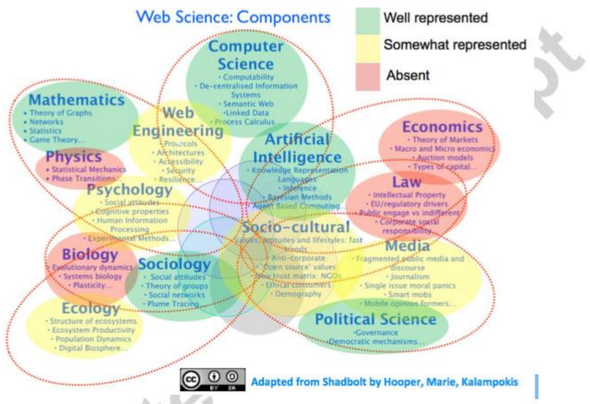


Figure 1. Web Science 'heat map', showing discipline presence [HM12]

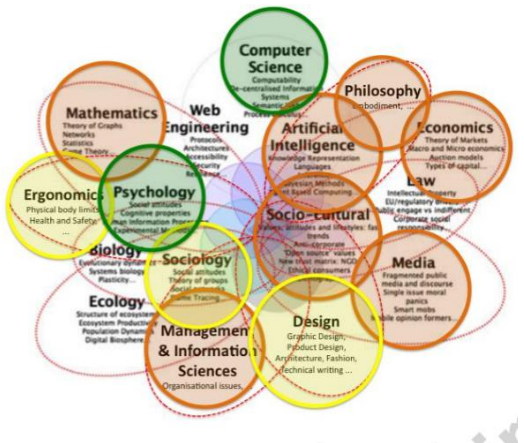


Figure 2. Heat map for HCI (from [HD13])

Borrowings from other fields have been very powerful to enable both theoretical and practical interventions. For example, Fitts' Law [Fi54] has created its own small sub-community, human–human conversation analysis has been used to design human–computer dialogues

[FL90], and Csikszentmihályi's concept of Flow [Cz90] has proved influential in user experience design.

However, while HCI can draw on the methods and knowledge of related fields directly, there are limits to this for two reasons:

different concerns – The questions we ask in HCI are typically more applied and hence more complex in terms of interrelations than 'base' disciplines, notably psychology. For example, early studies of on-screen reading comprehension, or more recent comparisons of reading comprehension when holding a screen vs with hands on the table [BJ11]; while in many ways these could be seen as standard perceptual and cognitive psychology, the reasons for studying both were practical and unlikely to have arisen purely from a psychological interest.

integrating knowledge – Some concerns really cut across disciplines requiring theoretical or practical knowledge from multiple areas of study. For example work on how design affects behaviour [PV15], requires both behavioural psychology and interaction design.

However, while there is copious empirical work of this kind, it is harder to find truly integrative HCI theory. There were early descriptive accounts, notably Norman's seven stages of action [No86,No88], and more predictive modelling approaches such as Card, Moran and Newells' 'Model Human Processor' [CM86] and Barnard's 'Interacting Cognitive Subsystems' [Ba85], but, while the former is still influential, there is no clear path of deepening theory of interaction.

Liu et al. [LG14] used co-word analysis techniques to investigate the development of themes and bodies of knowledge in HCI. This showed positive things, notably the level of cross cutting integration across the field; that is there are no disconnected camps. It also gave evidence to known effects such as the way that new technologies seem to buffet the discipline starting new themes, which rise for a period before trailing off. However, it also exposed a concerning dearth of integrating bodies of knowledge:

"As it stands, the only tradition in HCI is that of having no tradition in terms of research topics. … when a new technology comes along it seems that researchers start from scratch leading to relatively isolated research themes" [LG14].

Human – technology interaction One of the more recent changes in HCI is that computers really have become ubiquitous to the extent that it is rare to find any technology that does not involve computation, and if not in the artefact, in the design process, ordering, or manuals. This process began some time ago. In the mid 1990s I met appliance designers attending HCI conferences because they were beginning to have computer control panels and so they needed to understand interaction design principles. The difference is that at that stage computers were just beginning to be embedded in domestic technology, whereas now the two are almost synonymous, from heating radiators you can control from your mobile phone to QR codes on paper posters.

*So human–computer interaction is now effectively human–technology interaction.*

This creates new challenges for the discipline, but also opens up a longer history of human innovation and evolution of technology. That is in understating the foundations of HCI we can draw on millennia, not just the mere thirty to fifty years of digital development (rich and rapid that it has been).

## Graphical User Interface Design

User interface consists of everything in the system that people come into contact with, whether that is physically, perceptually or conceptually.

**Physically** people interact with systems in many different ways, such as by pressing buttons, touching a screen, moving a mouse over a table so that it moves a cursor over the screen, clicking a mouse button, rolling their thumb over a scroll wheel. We also interact physically through other senses, notably sound and touch.

**Perceptually** people interact with a system through what they can see, hear and touch. The visual aspects of interface design concern designing so that people will see and notice things on a screen. Buttons need to be big enough to see and they need to be labelled in a way that is understandable for people. Instructions need to be given so people know what they are expected to do. Displays of large amounts of information need to be carefully considered so that people can see the relationships between data to understand its significance.

**Conceptually** people interact with systems and devices through knowing what they can do and knowing how they can do it. Conceptually people employ a 'mental model' of what the device is and how it works. People need to know that certain commands exist that will allow them to do things. They need to know that certain data is available and the form that that data takes. They need to find their way to particular pieces of information (undertake navigation). They need to be able to find details of things, see an overview of things and focus on particular areas. Putting these three aspects together is the skill of the interface designer. Interface design is about creating an experience that enables people to make the best use of the system being designed. When first using a system people may well think 'right, I need to do X so I am going to use this device which means I am going to have to press Y on this keyboard and then press Z', but very soon people will form their intentions in the context of knowing what the system or device does and how they can achieve their goals. Physical, perceptual and conceptual design get woven together into the experiences of people.

## WIMPs

The most prevalent of the GUIs is the WIMP interface such as Windows or OS X. WIMP stands for windows, icons, menus and pointers. A window is a means of sharing a device's graphical display resources among multiple applications at the same time. An icon is an image or symbol used to represent a file, folder, application or device, such as a printer. David Canfield Smith is usually credited with coining the term in the context of user interfaces in 1975, while he worked at Xerox. According to Smith, he adopted the term from the Russian Orthodox Church where an icon is a religious image. A menu is a list of commands or options from which one can choose. The last component is a pointing device of which the mouse is the most widespread, but fingers are also used, as is the stylus. An important aspect of a WIMP environment is the manner in which we use it. This form of interaction is called direct manipulation because we directly manipulate the on-screen objects as opposed to issuing commands through a command-based interface.

**Every user interface**—whether it is designed for a WebApp, a traditional software application, a consumer product, or an industrial device— should exhibit the following characteristics: easy to use, easy to learn, easy to navigate, intuitive, consistent, efficient, errorfree, and functional. It should provide the end user with a satisfying and rewarding experience. Interface design concepts, principles, and methods provide a Web engineer with the tools required to achieve this list of attributes.

The user interface of a WebApp is its "fi rst impression." Regardless of the value of its content, the sophistication of its processing capabilities and services, and the overall benefit of the WebApp itself, a poorly designed interface will disappoint the potential user and may, in fact, cause the user to go elsewhere. Because of the sheer volume of competing WebApps in virtually every subject area, the interface must grab a potential user immediately. Bruce Tognozzi [Tog01] defines a set of fundamental characteristics that all interfaces should exhibit and, in doing so, establishes a philosophy that should be followed by every WebApp interface designer:

Effective interfaces are visually apparent and forgiving, instilling in their users a sense of control. Users quickly see the breadth of their options, grasp how to achieve their goals, and do their work. Effective interfaces do not concern the user with the inner workings of the system. Work is carefully and continuously saved, with full option for the user to undo any activity at any time. Effective applications and services perform a maximum of work, while requiring a minimum of information from users.

You might argue that everybody recognizes these things and every interface designer wants to achieve them. And yet, each of us has seen more than a few really bad interfaces and, surely, will continue to see many more.

**The Three Golden Rule**

1. Place the user in control.

2. Reduce the user's memory load.

3. Make the interface consistent.

These golden rules actually form the basis for a set of user interface design principles that guide this important aspect of software design.

**Place the User in Control**

During a requirements-gathering session for a major new information system, a key user was asked about the attributes of the window-oriented graphical interface. "What I really would like," said the user solemnly, "is a system that reads my mind. It knows what I want to do before I need to do it and makes it very easy for me to get it done. That's all, just that." My first reaction was to shake my head and smile, but I paused for a moment. There was absolutely nothing wrong with the user's request. She wanted a system that reacted to her needs and helped her get things done. She wanted to control the computer, not have the computer control her. Most interface constraints and restrictions that are imposed by a designer are intended to simplify the mode of interaction. But for whom? As a designer, you may be tempted to introduce constraints and limitations to simplify the implementation of the interface. The result may be an interface that is easy to build, but frustrating to use. Mandel [Man97] defines a number of design principles that allow the user to maintain control:

Define interaction modes in a way that does not force a user into unnecessary or undesired actions. An interaction mode is the current state of the interface. For example, if spell check is selected in a word-processor menu, the software moves to a spell-checking mode. There is no reason to force the user to remain in spell-checking mode if the user desires to make a small text edit along the way. The user should be able to enter and exit the mode with little or no effort.

Provide for flexible interaction. Because different users have different interaction preferences, choices should be provided. For example, software might allow a user to interact via keyboard commands, mouse movement, a digitizer pen, a multitouch screen, or voice recognition commands. But every action is not amenable to every interaction mechanism. Consider, for example, the difficulty of using keyboard command (or voice input) to draw a complex shape.

Allow user interaction to be interruptible and undoable. Even when involved in a sequence of actions, the user should be able to interrupt the sequence to do something else (without losing the work that had been done). The user should also be able to "undo" any action.

Streamline interaction as skill levels advance and allow the interaction to be customized. Users often find that they perform the same sequence of interactions repeatedly. It is worthwhile to design a "macro" mechanism that enables an advanced user to customize the interface to facilitate interaction.

Hide technical internals from the casual user. The user interface should move the user into the virtual world of the application. The user should not be aware of the operating system, file management functions, or other arcane computing technology. In essence, the interface should never require that the user interact at a level that is "inside" the machine (e.g., a user should never be required to type operating system commands from within application software).

Design for direct interaction with objects that appear on the screen. The user feels a sense of control when able to manipulate the objects that are necessary to perform a task in a manner similar to what would occur if the object were a physical thing. For example, an application interface that allows a user to "stretch" an object (scale it in size) is an implementation of direct manipulation.

**Reduce the User's Memory Load**

The more a user has to remember, the more error-prone the interaction with the system will be. It is for this reason that a well-designed user interface does not tax the user's memory. Whenever possible, the system should "remember" pertinent information and assist the user with an interaction scenario that assists recall. Mandel [Man97] defines design principles that enable an interface to reduce the user's memory load:

Reduce demand on short-term memory. When users are involved in complex tasks, the demand on short-term memory can be significant. The interface should be designed to reduce the requirement to remember past actions, inputs, and results. This can be accomplished by providing visual cues that enable a user to recognize past actions, rather than having to recall them.

Establish meaningful defaults. The initial set of defaults should make sense for the average user, but a user should be able to specify individual preferences. However, a "reset" option should be available, enabling the redefinition of original default values.

Define shortcuts that are intuitive. When mnemonics are used to accomplish a system function (e.g., alt-P to invoke the print function), the mnemonic should be tied to the action in a way that is easy to remember (e.g., first letter of the task to be invoked).

The visual layout of the interface should be based on a real-world metaphor. For example, a bill payment system should use a checkbook and check register metaphor to guide the user through the bill paying process. This enables the user to rely on well-understood visual cues, rather than memorizing an arcane interaction sequence.

Disclose information in a progressive fashion. The interface should be organized hierarchically. That is, information about a task, an object, or some behavior should be presented first at a high level of abstraction. More detail should be presented after the user indicates interest with a mouse pick. An example, common to many word-processing applications, is the underlining function. The function itself is one of a number of functions under a text style menu. However, every underlining capability is not listed. The user must pick underlining; then all underlining options (e.g., single underline, double underline, dashed underline) are presented.

### Make the Interface Consistent

The interface should present and acquire information in a consistent fashion. This implies that (1) all visual information is organized according to design rules that are maintained throughout all screen displays, (2) input mechanisms are constrained to a limited set that is used consistently throughout the application, and (3) mechanisms for navigating from task to task are consistently defined and implemented. Mandel [Man97] defines a set of design principles that help make the interface consistent:

Allow the user to put the current task into a meaningful context. Many interfaces implement complex layers of interactions with dozens of screen images. It is important to provide indicators (e.g., window titles, graphical icons, consistent color coding) that enable the user to know the context of the work at hand. In addition, the user should be able to determine where he has come from and what alternatives exist for a transition to a new task.

Maintain consistency across a family of applications. A set of applications (or products) should all implement the same design rules so that consistency is maintained for all interaction.

If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so. Once a particular interactive sequence has become a de facto standard (e.g., the use of alt-S to save a file), the user expects this in every application he encounters. A change (e.g., using alt-S to invoke scaling) will cause confusion.

## User Interface Analysis and Design

The overall process for analyzing and designing a user interface begins with the creation of different models of system function (as perceived from the outside). You begin by delineating the human- and computer-oriented tasks that are required to achieve system function and then considering the design issues that apply to all interface designs. Tools are used to prototype and ultimately implement the design model, and the result is evaluated by end users for quality.

### Interface Analysis and Design Models

Four different models come into play when a user interface is to be analyzed and designed. A human engineer (or the software engineer) establishes a user model, the software engineer creates a design model, the end user develops a mental image that is often called the user's mental model or the system perception, and the implementers of the system create an implementation model. Unfortunately, each of these models may differ significantly. Your role, as an interface designer, is to reconcile these differences and derive a consistent representation of the interface. The user model establishes the profile of end users of the system. In his introductory column on "user-centric design," Jeff Patton [Pat07] notes:

The truth is, designers and developers—myself included—often think about users. However, in the absence of a strong mental model of specific users, we self-substitute. Selfsubstitution isn't user centric—it's self-centric.

To build an effective user interface, "all design should begin with an understanding of the intended users, including profiles of their age, gender, physical abilities, education, cultural or ethnic background, motivation, goals and personality" [Shn04]. In addition, users can be categorized as:

Novices. No syntactic knowledge1 of the system and little semantic knowledge2 of the application or computer usage in general. Knowledgeable, intermittent users. Reasonable semantic knowledge of the application but relatively low recall of syntactic information necessary to use the interface. Knowledgeable, frequent users. Good semantic and syntactic knowledge that often leads to the "power-user syndrome"; that is, individuals who look for shortcuts and abbreviated modes of interaction.

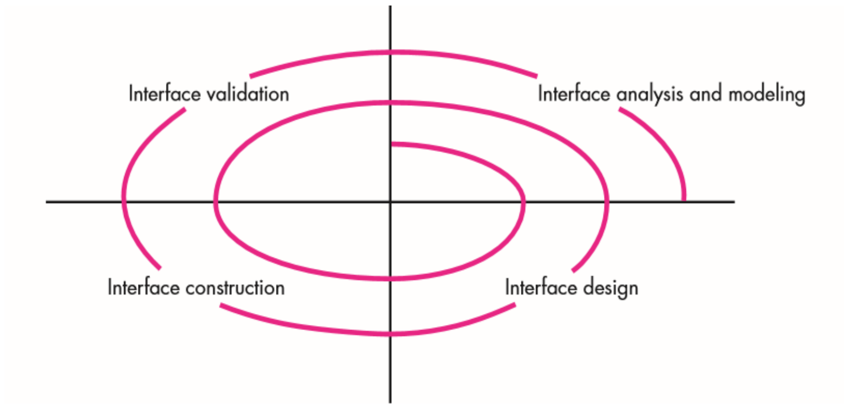
The user's mental model (system perception) is the image of the system that end users carry in their heads. For example, if the user of a particular word processor were asked to describe its operation, the system perception would guide the response. The accuracy of the description will depend upon the user's profile (e.g., novices would provide a sketchy response at best) and overall familiarity with software in the application domain. A user who understands word processors fully but has worked with the specific word processor only once might actually be able to provide a more complete description of its function than the novice who has spent weeks trying to learn the system. The implementation model combines the outward manifestation of the computer based system (the look and feel of the interface), coupled with all supporting information (books, manuals, videotapes, help files) that describes interface syntax and semantics. When the implementation model and the user's mental model are coincident, users generally feel comfortable with the software and use it effectively. To accomplish this "melding" of the models, the design model must have been developed to accommodate the information contained in the user model, and the implementation model must accurately reflect syntactic and semantic information about the interface. The models described in this section are "abstractions of what the user is doing or thinks he is doing or what somebody else thinks he ought to be doing when he uses an interactive system" [Mon84]. In essence, these models enable the interface designer to satisfy a key element of the most important principle of user interface design: "Know the user, know the tasks."

## The Process

The analysis and design process for user interfaces is iterative and can be represented using a spiral model. Referring to Figure 11.1, the user interface analysis and design process begins at the interior of the spiral and encompasses four distinct framework activities [Man97]: (1)

interface analysis and modeling, (2) interface design, (3) interface construction, and (4) interface validation. The spiral model implies that each of these tasks will occur more than once, with each pass around the spiral representing additional elaboration of requirements and the resultant design. In most cases, the construction activity involves prototyping—the only practical way to validate what has been designed. Interface analysis focuses on the profile of the users who will interact with the system. Skill level, business understanding, and general receptiveness to the new system are recorded; and different user categories are defined. For each user category, requirements are elicited. In essence, you work to understand the system perception for each class of users. Once general requirements have been defined, a more detailed task analysis is conducted. Those tasks that the user performs to accomplish the goals of the system

**The user interface design process**



are identified, described, and elaborated (over a number of iterative passes through the spiral). Finally, analysis of the user environment focuses on the physical work environment. Among the questions to be asked are:

● Where will the interface be located physically?

● Will the user be sitting, standing, or performing other tasks unrelated to the interface?

● Does the interface hardware accommodate space, light, or noise constraints?

● Are there special human factors considerations driven by environmental factors?

The information gathered as part of the analysis action is used to create an analysis model for the interface. Using this model as a basis, the design action commences. The goal of interface design is to define a set of interface objects and actions (and their screen representations) that enable a user to perform all defined tasks in a manner that meets every usability goal defined for the system. Interface construction normally begins with the creation of a prototype that enables usage scenarios to be evaluated. As the iterative design process continues, a user interface tool kit may be used to complete the construction of the interface. Interface validation focuses on (1) the ability of the interface to implement every user task correctly, to accommodate all task variations, and to achieve all general user requirements; (2) the degree to which the interface is easy to use and easy to learn, and (3) the users' acceptance of the interface as a useful tool in their work.

As I have already noted, the activities described in this section occur iteratively. Therefore, there is no need to attempt to specify every detail (for the analysis or design model) on the first pass. Subsequent passes through the process elaborate task detail, design information, and the operational features of the interface.

## Interface Analysis

A key tenet of all software engineering process models is this: understand the problem before you attempt to design a solution. In the case of user interface design, understanding the problem means understanding (1) the people (end users) who will interact with the system through the interface, (2) the tasks that end users must perform to do their work, (3) the content that is presented as part of the interface, and (4) the environment in which these tasks will be conducted. In the sections that follow, I examine each of these elements of interface analysis with the intent of establishing a solid foundation for the design tasks that follow.

User Analysis The phrase "user interface" is probably all the justification needed to spend some time understanding the user before worrying about technical matters. Earlier I noted that each user has a mental image of the software that may be different from the mental image developed by other users. In addition, the user's mental image may be vastly different from the software engineer's design model. The only way that you can get the mental image and the design model to converge is to work to understand the users themselves as well as how these people will use the system. Information from a broad array of sources can be used to accomplish this: User Interviews. The most direct approach, members of the software team meet with end users to better understand their needs, motivations, work culture, and a myriad of other issues. This can be accomplished in one-on-one meetings or through focus groups. Sales input. Sales people meet with users on a regular basis and can gather information that will help the software team to categorize users and better understand their requirements. Marketing input. Market analysis can be invaluable in the definition of market segments and an understanding of how each segment might use the software in subtly different ways. Support input. Support staff talks with users on a daily basis. They are the most likely source of information on what works and what doesn't, what users like and what they dislike, what features generate questions and what features are easy to use.

The following set of questions (adapted from [Hac98]) will help you to better understand the users of a system:

● Are users trained professionals, technicians, clerical, or manufacturing workers?

● What level of formal education does the average user have?

● Are the users capable of learning from written materials or have they expressed a desire for classroom training?

● Are users expert typists or keyboard phobic?

● What is the age range of the user community?

● Will the users be represented predominately by one gender?

● How are users compensated for the work they perform?

● Do users work normal office hours or do they work until the job is done?

● Is the software to be an integral part of the work users do or will it be used only occasionally?
● What is the primary spoken language among users?

● What are the consequences if a user makes a mistake using the system?

● Are users experts in the subject matter that is addressed by the system?

● Do users want to know about the technology that sits behind the interface? Once these questions are answered, you'll know who the end users are, what is likely to motivate and please them, how they can be grouped into different user classes or profiles, what their mental models of the system are, and how the user interface must be characterized to meet their needs.

Task Analysis and Modeling The goal of task analysis is to answer the following questions:

● What work will the user perform in specific circumstances?

● What tasks and subtasks will be performed as the user does the work?

● What specific problem domain objects will the user manipulate as work is performed?

● What is the sequence of work tasks—the workflow?

● What is the hierarchy of tasks? To answer these questions, you must draw upon techniques that I have discussed earlier in this book, but in this instance, these techniques are applied to the user interface.

Use cases. In earlier chapters you learned that the use case describes the manner in which an actor (in the context of user interface design, an actor is always a person) interacts with a system. When used as part of task analysis, the use case is developed to show how an end user performs some specific work-related task. In most instances, the use case is written in an informal style (a simple paragraph) in the first-person. For example, assume that a small software company wants to build a computer-aided design system explicitly for interior designers. To get a better understanding of how they do their work, actual interior designers are asked to describe a specific design function.

This use case provides a basic description of one important work task for the computer-aided design system. From it, you can extract tasks, objects, and the overall flow of the interaction. In addition, other features of the system that would please the interior designer might also be conceived. For example, a digital photo could be taken looking out each window in a room. When the room is rendered, the actual outside view could be represented through each window.

Task elaboration. stepwise elaboration (also called functional decomposition or stepwise refinement) is a mechanism for refining the processing tasks that are required for software to accomplish some desired function. Task analysis for interface design uses an elaborative approach to assist in understanding the human activities the user interface must accommodate. Task analysis can be applied in two ways. As I have already noted, an interactive, computer-based system is often used to replace a manual or semimanual activity. To understand the tasks that must be performed to accomplish the goal of the activity, you must understand the tasks that people currently perform (when using a manual approach) and then map these into a similar (but not necessarily identical) set of tasks that are implemented in the context of the user interface. Alternatively, you can study an existing specification for a computer-based solution and derive a set of user tasks that will accommodate the user model, the design model, and the system perception.

Regardless of the overall approach to task analysis, you must first define and classify tasks. I have already noted that one approach is stepwise elaboration. For example, let's reconsider the computer-aided design system for interior designers discussed earlier. By observing an interior designer at work, you notice that interior design comprises a number of major activities: furniture layout (note the use case discussed earlier), fabric and material selection, wall and window coverings selection, presentation (to the customer), costing, and shopping. Each of these major tasks can be elaborated into subtasks. For example, using information contained in the use case, furniture layout can be refined into the following tasks: (1) draw a floor plan based on room dimensions, (2) place windows and doors at appropriate locations, (3a) use furniture templates to draw scaled furniture outlines on the floor plan, (3b) use accents templates to draw scaled accents on the floor plan, (4) move furniture outlines and accent outlines to get the best placement, (5) label all furniture and accent outlines, (6) draw dimensions to show location, and (7) draw a perspective-rendering view for the customer. A similar approach could be used for each of the other major tasks.

Subtasks 1 to 7 can each be refined further. Subtasks 1 to 6 will be performed by manipulating information and performing actions within the user interface. On the other hand, subtask 7 can be performed automatically in software and will result in little direct user interaction.4 The design model of the interface should accommodate each of these tasks in a way that is consistent with the user model (the profile of a "typical" interior designer) and system perception (what the interior designer expects from an automated system).

Object elaboration. Rather than focusing on the tasks that a user must perform, you can examine the use case and other information obtained from the user and extract the physical objects that are used by the interior designer. These objects can be categorized into classes. Attributes of each class are defined, and an evaluation of the actions applied to each object provide a list of operations. For example, the furniture template might translate into a class called Furniture with attributes that might include size, shape, location, and others. The interior designer would select the object from the Furniture class, move it to a position on the floor plan (another object in this context), draw the furniture outline, and so forth. The tasks select, move, and draw are operations. The user interface analysis model would not provide a literal implementation for each of these operations. However, as the design is elaborated, the details of each operation are defined.

Workflow analysis. When a number of different users, each playing different roles, makes use of a user interface, it is sometimes necessary to go beyond task analysis and object elaboration and apply workflow analysis. This technique allows you to understand how a work process is completed when several people (and roles) are involved. Consider a company that intends to fully automate the process of prescribing and delivering prescription drugs. The entire process5 will revolve around a Web-based application that is accessible by physicians (or their assistants), pharmacists, and patients. Workflow can be represented effectively with a UML swimlane diagram (a variation on the activity diagram).

We consider only a small part of the work process: the situation that occurs when a patient asks for a refill. This information may have been elicited via interview or from use cases written by each actor. Regardless, the flow of events enables you to recognize a number of key interface characteristics: 1. Each user implements different tasks via the interface; therefore, the look and feel of the interface designed for the patient will be different than the one defined for pharmacists or physicians. 2. The interface design for pharmacists and physicians must accommodate access to and display of information from secondary information sources (e.g., access to inventory for the pharmacist and access to information about alternative medications for the physician). 3. Many of the activities noted in the swimlane diagram can be further elaborated using task analysis and/or object elaboration (e.g., Fills prescription could imply a mail-order delivery, a visit to a pharmacy, or a visit to a special drug distribution center).

Hierarchical representation. A process of elaboration occurs as you begin to analyze the interface. Once workflow has been established, a task hierarchy can be defined for each user type. The hierarchy is derived by a stepwise elaboration of each task identified for the user. For example, consider the following user task and subtask hierarchy.

User task: Requests that a prescription be refilled

● Provide identifying information.

● Specify name.

● Specify userid.

● Specify PIN and password.

● Specify prescription number.

● Specify date refill is required. To complete the task, three subtasks are defined. One of these subtasks, provide identifying information, is further elaborated in three additional sub-subtasks.

**Analysis of Display Content**

The user tasks identified lead to the presentation of a variety of different types of content. For modern applications, display content can range from character-based reports (e.g., a spreadsheet), graphical displays (e.g., a histogram, a 3-D model, a picture of a person), or specialized information (e.g., audio or video files). The analysis modeling techniques discussed in Chapters 6 and 7 identify the output data objects that are produced by an application. These data objects may be (1) generated by components (unrelated to the interface) in other parts of an application, (2) acquired from data stored in a database that is accessible from the application, or (3) transmitted from systems external to the application in question. During this interface analysis step, the format and aesthetics of the content (as it is displayed by the interface) are considered. Among the questions that are asked and answered are:

● Are different types of data assigned to consistent geographic locations on the screen (e.g., photos always appear in the upper right-hand corner)?

● Can the user customize the screen location for content?

● Is proper on-screen identification assigned to all content?

● If a large report is to be presented, how should it be partitioned for ease of understanding?

● Will mechanisms be available for moving directly to summary information for large collections of data?

● Will graphical output be scaled to fit within the bounds of the display device that is used?

● How will color be used to enhance understanding?

● How will error messages and warnings be presented to the user? The answers to these (and other) questions will help you to establish requirements for content presentation.

Hackos and Redish [Hac98] discuss the importance of work environment analysis when they state:

People do not perform their work in isolation. They are influenced by the activity around them, the physical characteristics of the workplace, the type of equipment they are using, and the work relationships they have with other people. If the products you design do not fit into the environment, they may be difficult or frustrating to use.

In some applications the user interface for a computer-based system is placed in a "user-friendly location" (e.g., proper lighting, good display height, easy keyboard access), but in others (e.g., a factory floor or an airplane cockpit), lighting may be suboptimal, noise may be a factor, a keyboard or mouse may not be an option, display placement may be less than ideal. The interface designer may be constrained by factors that mitigate against ease of use. In addition to physical environmental factors, the workplace culture also comes into play. Will system interaction be measured in some manner (e.g., time per transaction or accuracy of a transaction)? Will two or more people have to share information before an input can be provided? How will support be provided to users of the system? These and many related questions should be answered before the interface design commences.

## Interface Design Steps

Once interface analysis has been completed, all tasks (or objects and actions) required by the end user have been identified in detail and the interface design activity commences. Interface design, like all software engineering design, is an iterative process. Each user interface design step occurs a number of times, elaborating and refining information developed in the preceding step. Although many different user interface design models (e.g., [Nor86], [Nie00]) have been proposed, all suggest some combination of the following steps:

1. Using information developed during interface analysis (Section 11.3), define interface objects and actions (operations). 2. Define events (user actions) that will cause the state of the user interface to change. Model this behavior. 3. Depict each interface state as it will actually look to the end user. 4. Indicate how the user interprets the state of the system from information provided through the interface.

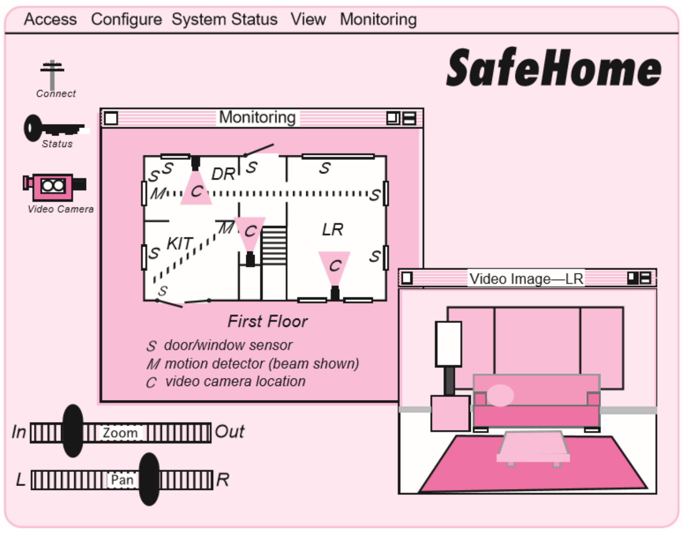
In some cases, you can begin with sketches of each interface state (i.e., what the user interface looks like under various circumstances) and then work backward to define objects, actions, and other important design information. Regardless of the sequence of design tasks, you should (1) always follow the golden rules, (2) model how the interface will be implemented, and (3) consider the environment (e.g., display technology, operating system, development tools) that will be used.

### Applying Interface Design Steps

The definition of interface objects and the actions that are applied to them is an important step in interface design. To accomplish this, user scenarios are parsed in much the same way as described in Chapter 6. That is, a use case is written. Nouns (objects) and verbs (actions) are isolated to create a list of objects and actions. Once the objects and actions have been defined and elaborated iteratively, they are categorized by type. Target, source, and application objects are identified. A source object (e.g., a report icon) is dragged and dropped onto a target object (e.g., a printer icon). The implication of this action is to create a hard-copy report. An application object represents application-specific data that are not directly manipulated as part of screen interaction. For example, a mailing list is used to store names for a mailing. The list itself might be sorted, merged, or purged (menu-based actions), but it is not dragged and dropped via user interaction. When you are satisfied that all important objects and actions have been defined (for one design iteration), screen layout is performed. Like other interface design activities, screen layout is an interactive process in which graphical design and placement of icons, definition of descriptive screen text, specification and titling for windows, and definition of major and minor menu items are conducted. If a real-world metaphor is appropriate for the application, it is specified at this time, and the layout is organized in a manner that complements the metaphor. To provide a brief illustration of the design steps noted previously, consider a user scenario for the SafeHome system (discussed in earlier chapters). A preliminary use case (written by the homeowner) for the interface follows:

Preliminary use case: I want to gain access to my SafeHome system from any remote location via the Internet. Using browser software operating on my notebook computer (while I'm at work or traveling), I can determine the status of the alarm system, arm or disarm the system, reconfigure security zones, and view different rooms within the house via preinstalled video cameras.

**Preliminary screen layout**



## Design Issues

As the design of a user interface evolves, four common design issues almost always surface: system response time, user help facilities, error information handling, and command labeling. Unfortunately, many designers do not address these issues until relatively late in the design process (sometimes the first inkling of a problem doesn't occur until an operational prototype is available). Unnecessary iteration, project delays, and end-user frustration often result. It is far better to establish each as a design issue to be considered at the beginning of software design, when changes are easy and costs are low.

Response time. System response time is the primary complaint for many interactive applications. In general, system response time is measured from the point at which the user performs some control action (e.g., hits the return key or clicks a mouse) until the software responds with desired output or action. System response time has two important characteristics: length and variability. If system response is too long, user frustration and stress are inevitable. Variability refers to the deviation from average response time, and in many ways, it is the most important response time characteristic. Low variability enables the user to establish an interaction rhythm, even if response time is relatively long. For example, a 1-second response to a command will often be preferable to a response that varies from 0.1 to 2.5 seconds. When variability is significant, the user is always off balance, always wondering whether something "different" has occurred behind the scenes.

Help facilities. Almost every user of an interactive, computer-based system requires help now and then. In some cases, a simple question addressed to a knowledgeable colleague can do the trick. In others, detailed research in a multivolume set of "user manuals" may be the only option. In most cases, however, modern software provides online help facilities that enable a user to get a question answered or resolve a problem without leaving the interface. A number of design issues [Rub88] must be addressed when a help facility is considered:

● Will help be available for all system functions and at all times during system interaction? Options include help for only a subset of all functions and actions or help for all functions. ● How will the user request help? Options include a help menu, a special function key, or a HELP command. ● How will help be represented? Options include a separate window, a reference to a printed document (less than ideal), or a one- or two-line suggestion produced in a fixed screen location. ● How will the user return to normal interaction? Options include a return button displayed on the screen, a function key, or control sequence.

How will help information be structured? Options include a "flat" structure in which all information is accessed through a keyword, a layered hierarchy of information that provides increasing detail as the user proceeds into the structure, or the use of hypertext.

Error handling. Error messages and warnings are "bad news" delivered to users of interactive systems when something has gone awry. At their worst, error messages and warnings impart useless or misleading information and serve only to increase user frustration. There are few computer users who have not encountered an error of the form: "Application XXX has been forced to quit because an error of type 1023 has been encountered." Somewhere, an explanation for error 1023 must exist; otherwise, why would the designers have added the identification? Yet, the error message provides no real indication of what went wrong or where to look to get additional information. An error message presented in this manner does nothing to assuage user anxiety or to help correct the problem. In general, every error message or warning produced by an interactive system should have the following characteristics:

● The message should describe the problem in jargon that the user can understand. ● The message should provide constructive advice for recovering from the error. ● The message should indicate any negative consequences of the error (e.g., potentially corrupted data files) so that the user can check to ensure that they have not occurred (or correct them if they have). ● The message should be accompanied by an audible or visual cue. That is, a beep might be generated to accompany the display of the message, or the message might flash momentarily or be displayed in a color that is easily recognizable as the "error color." ● The message should be "nonjudgmental." That is, the wording should never place blame on the user.

Because no one really likes bad news, few users will like an error message no matter how well designed. But an effective error message philosophy can do much to improve the quality of an interactive system and will significantly reduce user frustration when problems do occur.

Menu and command labeling. The typed command was once the most common mode of interaction between user and system software and was commonly used for applications of every type. Today, the use of window-oriented, point-andpick interfaces has reduced reliance on typed commands, but some power-users continue to prefer a command-oriented mode of interaction. A number of design issues arise when typed commands or menu labels are provided as a mode of interaction: ● Will every menu option have a corresponding command? ● What form will commands take? Options include a control sequence (e.g., alt-P), function keys, or a typed word. ● How difficult will it be to learn and remember the commands? What can be done if a command is forgotten? ● Can commands be customized or abbreviated by the user? ● Are menu labels self-explanatory within the context of the interface? ● Are submenus consistent with the function implied by a master menu item? As I noted earlier in this chapter, conventions for command usage should be established across all applications. It is confusing and often error-prone for a user to type alt-D when a graphics object is to be duplicated in one application and alt-D when a graphics object is to be deleted in another. The potential for error is obvious.

Application accessibility. As computing applications become ubiquitous, software engineers must ensure that interface design encompasses mechanisms that enable easy access for those with special needs. Accessibility for users (and software engineers) who may be physically challenged is an imperative for ethical, legal, and business reasons. A variety of accessibility guidelines (e.g., [W3C03])—many designed for Web applications but often applicable to all types of software—provide detailed suggestions for designing interfaces that achieve varying levels of accessibility. Others (e.g., [App08], [Mic08]) provide specific guidelines for "assistive technology" that addresses the needs of those with visual, hearing, mobility, speech, and learning impairments.

Internationalization. Software engineers and their managers invariably underestimate the effort and skills required to create user interfaces that accommodate the needs of different locales and languages. Too often, interfaces are designed for one locale and language and then jury-rigged to work in other countries. The challenge for interface designers is to create "globalized" software. That is, user interfaces should be designed to accommodate a generic core of functionality that can be delivered to all who use the software. Localization features enable the interface to be customized for a specific market. A variety of internationalization guidelines (e.g., [IBM03]) are available to software engineers. These guidelines address broad design issues (e.g., screen layouts may differ in various markets) and discrete implementation issues (e.g., different alphabets may create specialized labeling and spacing requirements). The Unicode standard [Uni03] has been developed to address the daunting challenge of managing dozens of natural languages with hundreds of characters and symbols.

## Web Application Interface Design

Every user interface—whether it is designed for a WebApp, a traditional software application, a consumer product, or an industrial device—should exhibit the usability characteristics that were discussed earlier in this chapter. Dix [Dix99] argues that you should design a WebApp interface so that it answers three primary questions for the end user:

Where am I? The interface should (1) provide an indication of the WebApp that has been accessed8 and (2) inform the user of her location in the content hierarchy. What can I do now? The interface should always help the user understand his current options—what functions are available, what links are live, what content is relevant? Where have I been, where am I going? The interface must facilitate navigation. Hence, it must provide a "map" (implemented in a way that is easy to understand) of where the user has been and what paths may be taken to move elsewhere within the WebApp.

An effective WebApp interface must provide answers for each of these questions as the end user navigates through content and functionality.

**Interface Design Principles and Guidelines**

The user interface of a WebApp is its "first impression." Regardless of the value of its content, the sophistication of its processing capabilities and services, and the overall benefit of the WebApp itself, a poorly designed interface will disappoint the potential user and may, in fact, cause the user to go elsewhere. Because of the sheer volume of competing WebApps in virtually every subject area, the interface must "grab" a potential user immediately. Bruce Tognozzi [Tog01] defines a set of fundamental characteristics that all interfaces should exhibit and in doing so, establishes a philosophy that should be followed by every WebApp interface designer:

Effective interfaces are visually apparent and forgiving, instilling in their users a sense of control. Users quickly see the breadth of their options, grasp how to achieve their goals, and do their work. Effective interfaces do not concern the user with the inner workings of the system. Work is carefully and continuously saved, with full option for the user to undo any activity at any time. Effective applications and services perform a maximum of work, while requiring a minimum of information from users.

In order to design WebApp interfaces that exhibit these characteristics, Tognozzi [Tog01] identifies a set of overriding design principles:

**Anticipation**. A WebApp should be designed so that it anticipates the user's next move. For example, consider a customer support WebApp developed by a manufacturer of computer printers. A user has requested a content object that presents information about a printer driver for a newly released operating system. The designer of the WebApp should anticipate that the user might request a download of the driver and should provide navigation facilities that allow this to happen without requiring the user to search for this capability.

**Communication**. The interface should communicate the status of any activity initiated by the user. Communication can be obvious (e.g., a text message) or subtle (e.g., an image of a sheet of paper moving through a printer to indicate that printing is under way). The interface should also communicate user status (e.g., the user's identification) and her location within the WebApp content hierarchy.

**Consistency**. The use of navigation controls, menus, icons, and aesthetics (e.g., color, shape, layout) should be consistent throughout the WebApp.For example, if underlined blue text implies a navigation link, content should never incorporate blue underlined text that does not imply a link. In addition, an object, say a yellow triangle, used to indicate a caution message before the user invokes a particular function or action, should not be used for other purposes elsewhere in the WebApp. Finally, every feature of the interface should respond in a manner that is consistent with user expectations.

**Controlled autonomy**. The interface should facilitate user movement throughout the WebApp, but it should do so in a manner that enforces navigation conventions that have been established for the application. For example, navigation to secure portions of the WebApp should be controlled by userID and password, and there should be no navigation mechanism that enables a user to circumvent these controls.

**Efficiency**. The design of the WebApp and its interface should optimize the user's work efficiency, not the efficiency of the developer who designs and builds it or the client server environment that executes it. Tognozzi [Tog01] discusses this when he writes: "This simple truth is why it is so important for everyone involved in a software project to appreciate the importance of making user productivity goal one and to understand the vital difference between building an efficient system and empowering an efficient user."

**Flexibility**. The interface should be flexible enough to enable some users to accomplish tasks directly and others to explore the WebApp in a somewhat random fashion. In every case, it should enable the user to understand where he is and provide the user with functionality that can undo mistakes and retrace poorly chosen navigation paths.

**Focus**. The WebApp interface (and the content it presents) should stay focused on the user task(s) at hand. In all hypermedia there is a tendency to route the user to loosely related content. Why? Because it's very easy to do! The problem is that the user can rapidly become lost in many layers of supporting information and lose sight of the original content that she wanted in the first place.

**Fitt's law**. "The time to acquire a target is a function of the distance to and size of the target" [Tog01]. Based on a study conducted in the 1950s [Fit54], Fitt's law "is an effective method of modeling rapid, aimed movements, where one appendage (like a hand) starts at rest at a specific start position, and moves to rest within a target area" [Zha02]. If a sequence of selections or standardized inputs (with many different options within the sequence) is defined by a user task, the first selection (e.g., mouse pick) should be physically close to the next selection. For example, consider a WebApp home page interface at an e-commerce site that sells consumer electronics. Each user option implies a set of follow-on user choices or actions. For example, the "buy a product" option requires that the user enter a product category followed by the product name. The product category (e.g., audio equipment, televisions, DVD players) appears as a pull-down menu as soon as "buy a product" is picked. Therefore, the next choice is immediately obvious (it is nearby) and the time to acquire it is negligible. If, on the other hand, the choice appeared on a menu that was located on the other side of the screen, the time for the user to acquire it (and then make the choice) would be far too long.

**Human interface objects**. A vast library of reusable human interface objects has been developed for WebApps. Use them. Any interface object that can be "seen, heard, touched or otherwise perceived" [Tog01] by an end user can be acquired from any one of a number of object libraries.

**Latency reduction**. Rather than making the user wait for some internal operation to complete (e.g., downloading a complex graphical image), the WebApp should use multitasking in a way that lets the user proceed with work as if the operation has been completed.In addition to reducing latency, delays must be acknowledged so that the user understands what is happening. This includes (1) providing audio feedback when a selection does not result in an immediate action by the WebApp, (2) displaying an animated clock or progress bar to indicate that processing is under way, and (3) providing some entertainment (e.g., an animation or text presentation) while lengthy processing occurs.

**Learnability**. A WebApp interface should be designed to minimize learning time, and once learned, to minimize relearning required when the WebApp is revisited. In general the interface should emphasize a simple, intuitive design that organizes content and functionality into categories that are obvious to the user.

**Metaphors**. An interface that uses an interaction metaphor is easier to learn and easier to use, as long as the metaphor is appropriate for the application and the user. A metaphor should call on images and concepts from the user's experience, but it does not need to be an exact reproduction of a real-world experience. For example, an e-commerce site that implements automated bill paying for a financial institution, uses a checkbook metaphor (not surprisingly) to assist the user in specifying and scheduling bill payments. However, when a user "writes" a check, he need not enter the complete payee name but can pick from a list of payees or have the system select based on the first few typed letters. The metaphor remains intact, but the user gets an assist from the WebApp.

**Maintain work product integrity**. A work product (e.g., a form completed by the user, a user-specified list) must be automatically saved so that it will not be lost if an error occurs. Each of us has experienced the frustration associated with completing a lengthy WebApp form only to have the content lost because of an error (made by us, by the WebApp, or in transmission from client to server). To avoid this, a WebApp should be designed to autosave all user-specified data. The interface should support this function and provide the user with an easy mechanism for recovering "lost" information.

**Readability**. All information presented through the interface should be readable by young and old. The interface designer should emphasize readable type styles, font sizes, and color background choices that enhance contrast.

**Track state**. When appropriate, the state of the user interaction should be tracked and stored so that a user can logoff and return later to pick up where she left off. In general, cookies can be designed to store state information. However, cookies are a controversial technology, and other design solutions may be more palatable for some users.

**Visible navigation**. A well-designed WebApp interface provides "the illusion that users are in the same place, with the work brought to them" [Tog01]. When this approach is used, navigation is not a user concern. Rather, the user retrieves content objects and selects functions that are displayed and executed through the interface.

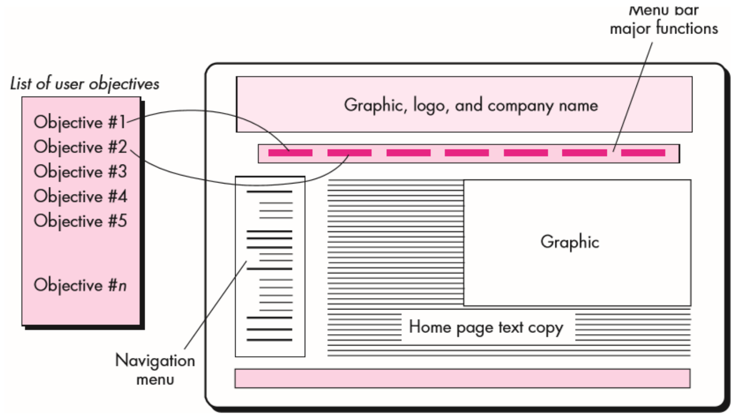
Nielsen and Wagner [Nie96] suggest a few pragmatic interface design guidelines (based on their redesign of a major WebApp) that provide a nice complement to the principles suggested earlier in this section: ● Reading speed on a computer monitor is approximately 25 percent slower than reading speed for hardcopy. Therefore, do not force the user to read voluminous amounts of text, particularly when the text explains the operation of the WebApp or assists in navigation. ● Avoid "under construction" signs—an unnecessary link is sure to disappoint. ● Users prefer not to scroll. Important information should be placed within the dimensions of a typical browser window. ● Navigation menus and head bars should be designed consistently and should be available on all pages that are available to the user. The design should not rely on browser functions to assist in navigation. ● Aesthetics should never supersede functionality. For example, a simple button might be a better navigation option than an aesthetically pleasing, but vague image or icon whose intent is unclear. ● Navigation options should be obvious, even to the casual user. The user should not have to search the screen to determine how to link to other content or services.

A well-designed interface improves the user's perception of the content or services provided by the site. It need not necessarily be flashy, but it should always be well structured and ergonomically sound.

### Interface Design Workflow for Web Applications

Earlier in this chapter I noted that user interface design begins with the identification of user, task, and environmental requirements. Once user tasks have been identified, user scenarios (use cases) are created and analyzed to define a set of interface objects and actions. Information contained within the requirements model forms the basis for the creation of a screen layout that depicts graphical design and placement of icons, definition of descriptive screen text, specification and titling for windows, and specification of major and minor menu items. Tools are then used to prototype and ultimately implement the interface design model. The following tasks represent a rudimentary workflow for WebApp interface design: 1. Review information contained in the requirements model and refine as required. 2. Develop a rough sketch of the WebApp interface layout. An interface prototype (including the layout) may have been developed as part of the requirements modeling activity. If the layout already exists, it should be reviewed and refined as required. If the interface layout has not been

**Mapping user objectives into interface actions**



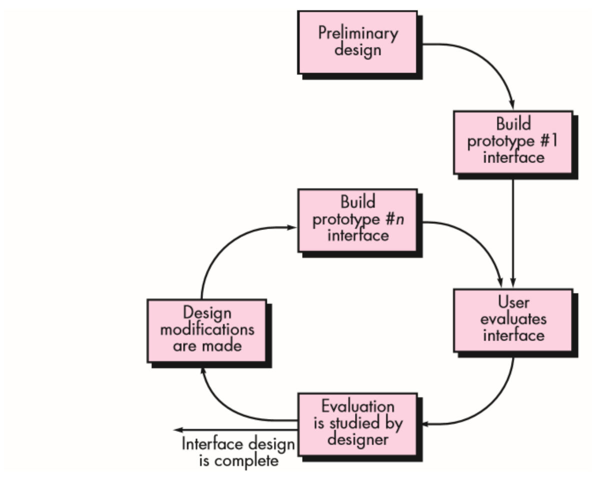
developed, you should work with stakeholders to develop it at this time. A schematic first-cut layout sketch is shown above. Map user objectives into specific interface actions. For the vast majority of WebApps, the user will have a relatively small set of primary objectives. These should be mapped into specific interface actions as shown. In essence, you must answer the following question: “How does the interface enable the user to accomplish each objective?” 4. Define a set of user tasks that are associated with each action. Each interface action (e.g., “buy a product”) is associated with a set of user tasks. These tasks have been identified during requirements modeling. During design, they must be mapped into specific interactions that encompass navigation issues, content objects, and WebApp functions. 5. Storyboard screen images for each interface action. As each action is considered, a sequence of storyboard images (screen images) should be created to depict how the interface responds to user interaction. Content objects should be identified (even if they have not yet been designed and developed), WebApp functionality should be shown, and navigation links should be indicated. 6. Refine interface layout and storyboards using input from aesthetic design. In most cases, you’ll be responsible for rough layout and storyboarding, but the aesthetic look and feel for a major commercial site is often developed by artistic, rather than technical, professionals. Aesthetic design is integrated with the work performed by the interface designer.

Identify user interface objects that are required to implement the interface. This task may require a search through an existing object library to find those reusable objects (classes) that are appropriate for the WebApp interface. In addition, any custom classes are specified at this time. 8. Develop a procedural representation of the user’s interaction with the interface. This optional task uses UML sequence diagrams and/or activity diagrams (Appendix 1) to depict the flow of activities (and decisions) that occur as the user interacts with the WebApp. 9. Develop a behavioral representation of the interface. This optional task makes use of UML state diagrams (Appendix 1) to represent state transitions and the events that cause them. Control mechanisms (i.e., the objects and actions available to the user to alter a WebApp state) are defined. 10. Describe the interface layout for each state. Using design information developed in Tasks 2 and 5, associate a specific layout or screen image with each WebApp state described in Task 8. 11. Refine and review the interface design model. Review of the interface should focus on usability.

It is important to note that the final task set you choose should be adapted to the special requirements of the application that is to be built.

## Design Evaluation Cycle

Once you create an operational user interface prototype, it must be evaluated to determine whether it meets the needs of the user. Evaluation can span a formality spectrum that ranges from an informal “test drive,” in which a user provides impromptu feedback to a formally designed study that uses statistical methods for the evaluation of questionnaires completed by a population of end users. The user interface evaluation cycle takes the form shown. After the design model has been completed, a first-level prototype is created. The prototype is evaluated by the user, who provides you with direct comments about the efficacy of the interface. In addition, if formal evaluation techniques are used (e.g., questionnaires, rating sheets), you can extract information from these data (e.g., 80 percent of all users did not like the mechanism for saving data files). Design modifications are made based on user input, and the next level prototype is created. The evaluation cycle continues until no further modifications to the interface design are necessary.


Preliminary design
Build prototype #1 interface
Build prototype #n interface
Design modifications are made
User evaluates interface
Evaluation is studied by designer
Interface design is complete

The interface design evaluation cycle

The prototyping approach is effective, but is it possible to evaluate the quality of a user interface before a prototype is built? If you identify and correct potential problems early, the number of loops through the evaluation cycle will be reduced and development time will shorten. If a design model of the interface has been created, a number of evaluation criteria [Mor81] can be applied during early design reviews:

1. The length and complexity of the requirements model or written specification of the system and its interface provide an indication of the amount of learning required by users of the system. 2. The number of user tasks specified and the average number of actions per task provide an indication of interaction time and the overall efficiency of the system. 3. The number of actions, tasks, and system states indicated by the design model imply the memory load on users of the system. 4. Interface style, help facilities, and error handling protocol provide a general indication of the complexity of the interface and the degree to which it will be accepted by the user.

Once the first prototype is built, you can collect a variety of qualitative and quantitative data that will assist in evaluating the interface. To collect qualitative data, questionnaires can be distributed to users of the prototype. Questions can be: (1) simple yes/no response, (2) numeric response, (3) scaled (subjective) response, (4) Likert scales (e.g., strongly agree, somewhat agree), (5) percentage (subjective) response, or (6) open-ended. If quantitative data are desired, a form of time-study analysis can be conducted. Users are observed during interaction, and data—such as number of tasks correctly completed over a standard time period, frequency of actions, sequence of actions, time spent "looking" at the display, number and types of errors, error recovery time, time spent using help, and number of help references per standard time period—are collected and used as a guide for interface modification.

**Voice User Interface**

Voice user interfaces (VUIs) allow the user to interact with a system through voice or speech commands. Virtual assistants, such as Siri, Google Assistant, and Alexa, are examples of VUIs. The primary advantage of a VUI is that it allows for a hands-free, eyes-free way in which users can interact with a product while focusing their attention elsewhere.

Applying the same design guidelines to VUIs as to graphical user interfaces is impossible. In a VUI, there *are* no visual affordances; so, when looking at a VUI, users have *no* clear indications of what the interface can do or what their options are. When designing VUI actions, it is thus important that the system clearly state possible interaction options, tell the user what

functionality he/she is using, and limit the amount of information it gives out to an amount that users can remember.
Because individuals normally associate voice with interpersonal communication rather than with person-technology interaction, they are sometimes unsure of the complexity to which the VUI can understand. Hence, for a VUI to succeed, it not only requires an excellent ability to understand spoken language but also needs to train users to understand what type of voice commands they can use and what type of interactions they can perform. The intricate nature of a user's conversing with a VUI means a designer needs to pay close attention to how easily a user might overstep with expectations. This is why designing the product in such a simple, almost featureless form is important—to keep the user mindful that a two-way "human" conversation is infeasible. Likewise, the user's patience in building a communications "rapport" will help improve satisfaction when the VUI, becoming increasingly acquainted with the speaker's voice (which the speaker will use more effectively), rewards him/her with more accurate responses.

**Design principle for interactive systems**

Over the years many principles of good interactive system design have been developed. Don Norman in his book The Design of Everyday Things (Norman, 1998) provides several, as does Jacob Nielsen in Usability Engineering (Nielsen, 1993). However, the level of abstraction provided by different people at different times is sometimes rather inconsistent and confusing. Design principles can be very broad or they can be more specific. There are also good design principles that derive from psychology, such as 'minimize memory load', i.e. do not expect people to remember too much. Apple, Microsoft and Google all provide user interface design guidelines for the development of products that run on their platforms.

The application of design principles has led to established design guidelines and patterns of interaction in certain circumstances such as the 'Undo' command in a Windows application, the 'Back' button on a website or the greying-out of inappropriate options on menus. Design principles can guide the designer during the design process and can be used to evaluate and critique prototype design ideas. Our list of high-level design principles, put together from Norman, Nielsen and others, is shown below. All the principles interact in complex ways, affecting each other, sometimes conflicting with each other and sometimes enhancing each other. But they help to orientate the designer to key features of good design and sensitize the designer to important issues. For ease of memorizing and use we have grouped them into three main categories - learnability, effectiveness and accommodation - but these groupings are not rigid. Systems should be leamable, effective and accommodating.

● Principles 1-4 are concerned with access, ease of learning and remembering (learnability).

● Principles 5-7 are concerned with ease of use, and principles 8 and 9 with safety (effectiveness).

● Principles 10-12 are concerned with accommodating differences between people and respecting those differences (accommodation).

Designing interactive systems from a human-centred perspective is concerned with the following.

Helping people access, learn and remember the system

1 **Visibility**. Try to ensure that things are visible so that people can see what functions are available and what the system is currently doing. This is an important part of the psychological principle that it is easier to recognize things than to have to recall them. If it is not possible to make it visible, make it observable. Consider making things Visible' through the use of sound and touch.

2 **Consistency**. Be consistent in the use of design features and be consistent with similar systems and standard ways of working. Consistency can be something of a slippery concept (see the Further thoughts box). Both conceptual and physical consistency are important.

3 **Familiarity**. Use language and symbols that the intended audience will be familiar with. Where this is not possible because the concepts are quite different from those people know about, provide a suitable metaphor to help them transfer similar and related knowledge from a more familiar domain.

4 **Affordance**. Design things so it is clear what they are for; for example, make buttons look like push buttons so people will press them. Affordance refers to the properties that things have (or are perceived to have) and how these relate to how the things could be used. Buttons afford pressing, chairs afford sitting on, and Post-it notes afford writing a message on and sticking next to something else. Affordances are culturally determined.

Giving them the seme of being in control, knowing what to do and how to do it

5 **Navigation**. Provide support to enable people to move around the parts of the system: maps, directional signs and information signs.

6 **Control**. Make it clear who or what is in control and allow people to take control. Control is enhanced if there is a clear, logical mapping between controls and the effect that they have. Also make clear the relationship between what the system does and what will happen in the world outside the system.

7 **Feedback**. Rapidly feed back information from the system to people so that they know what effect their actions have had. Constant and consistent feedback will enhance the feeling of control.

Safely and securely

8 **Recovery**. Enable recovery from actions, particularly mistakes and errors, quickly and effectively.

9 **Constraints**. Provide constraints so that people do not try to do things that are inappropriate. In particular, people should be prevented from making serious errors through properly constraining allowable actions and seeking confirmation of dangerous operations.

In a way that suits them

10 **Flexibility**. Allow multiple ways of doing things so as to accommodate people with different levels of experience and interest in the system. Provide people with the opportunity to change the way things look or behave so that they can personalize the system.

11 **Style**. Designs should be stylish and attractive.

12 **Conviviality**. Interactive systems should be polite, friendly and generally pleasant. Nothing ruins the experience of using an interactive system more than an aggressive message or an abrupt interruption. Design for politeness. Conviviality also suggests joining in and using interactive technologies to connect and support people.

## The Web

Today, we're living in the formative years of the Internet era. So much has already been said about this exciting time that it's impossible to discuss the impact of the Internet and the World Wide Web without lapsing into a cliché-ridden dialogue. You already know the Web is big, very big. But we don't mean "big" in the typical sense (e.g., number of Web pages and sites, number of users, amount of information flowing across the network), although the size of the Web and its projected growth rate are staggering. We mean big in a societal and cultural sense.

The Web has become an indispensable technology for business, commerce, communication, education, engineering, entertainment, finance, government, industry, media, medicine, politics, science, and transportation—to name just a few areas that impact your life. But being an "indispensable technology" only scratches the surface of the Web's impact on each of us. It has changed the ways in which we buy products (e-commerce), meet people (online dating), understand the world (portals), acquire our news (online media), voice our opinions (blogs), entertain ourselves (everything from music downloads to online casinos), and go to school (online learning).

All of these impacts have one thing in common—they need a delivery vehicle that takes raw information associated with the area of interest; structures it in a way that is meaningful; builds a packaged presentation that is organized, aesthetic, ergonomic, and interactive (where required); and delivers it to your Web browser in a manner that initiates a conversation. The conversation between you and a Web application can be passive or active. In a passive conversation, you select the information that is to be presented, but have no direct control over its volume, type, or structure. In an active conversation, you provide input so that the information that is presented is customized to meet your needs specifically. The vehicle that acquires information, structures it, builds a packaged presentation, and delivers it is called a Web application (WebApp). When a Web App is combined with client and server hardware, operating systems, network software, and browsers, a Web-based system emerges.

## Web accessibility

The Internet has become an important source of information and communication all over the world. Internet statistics show that there are more than 2 billion internet users in the world [1]. Websites and web based user interface for applications are becoming very popular as a means of user interaction and spread of information among differently abled people too. World Health Organization reports that there are approximately 785 million people who live with disabilities in the world [2].

## What is accessibility and how does it apply to interface design?

As Web Applications become ubiquitous, Web engineers must ensure that interface design encompasses mechanisms that enable easy access for those with special needs. Accessibility for users (and developers) who may be physically challenged is an imperative for moral, legal, and business reasons. A variety of accessibility guidelines (e.g., [W3C03]) provide detailed suggestions for designing interfaces that achieve varying levels of accessibility. Others [e.g., [App07 (*Apple Inc., Accessibility, 2007, www.apple.com/accessibility*)] and [*Mic07 (Microsoft, Accessibility, 2007, www.microsoft.com/enable*)]) provide specific guidelines for "assistive technology" that addresses the needs of those with visual, hearing, mobility, speech, and learning impairments. Guidelines for developing accessible software can also be found at the IBM Human Ability and Accessibility Center, www-03.ibm.com/able/access_ibm/disability.html.

Disabled people use assistive technologies, a term used to refer to hardware and software designed to facilitate the use of computers by people with disabilities (DRC 2004), to access the Web (see Assistive Technologies). These technologies work satisfactorily as long as the page is designed well. However, this is not the case for many pages (Takagi et al. 2007).

The W3C Web Accessibility Initiative (WAI) recognises this and provides guidelines to promote accessibility on the Web including Web Content Accessibility Guidelines (WCAG 1.0) (Chisholm et al. 1999). While many organisations such as the RNIB (Royal National Institute of Blind People) also highlight some of the important accessibility issues, the W3C accessibility guidelines are more complete and cover the key points of all the others. Besides WCAG 1.0, the W3C also provides guidelines for user agents and authoring tools. However, as recent surveys demonstrate (DRC), few designers follow these guidelines. Unfortunately, all attempts which have focused on guidelines have failed to give unilateral accessibility across the board because they are optional, not enforceable, and not accurately testable. While version two of the WCAG guidelines (not yet ratified by the W3C) promises to be more testable, with suggested routes to validation built in at the start, the guidelines still cover over 200 pages with an additional 200 page 'how-to' annex. In this case, it would seem that only the most dedicated designer will know enough to design to accessibility standards and produce pages which validate correctly. Indeed, because Web browsers present content on the screen, the visual rendering is often the only 'validity' check a designer or author performs. However, this check is flawed as most browsers attempt to correct badly written, and inaccessible, code as it is displayed. Therefore, although Web guidelines direct designers and authors to best practice, currently, most Web sites have accessibility barriers that make it either difficult or near impossible for many people with disabilities to use these sites. There are also evaluation, validation, and repair tools (see Web Accessibility Evaluation) to check Web pages against best practice and guidelines. In brief, validation and repair tools analyse pages against accessibility guidelines and return a report or a rating (Ivory and Hearst 2001).

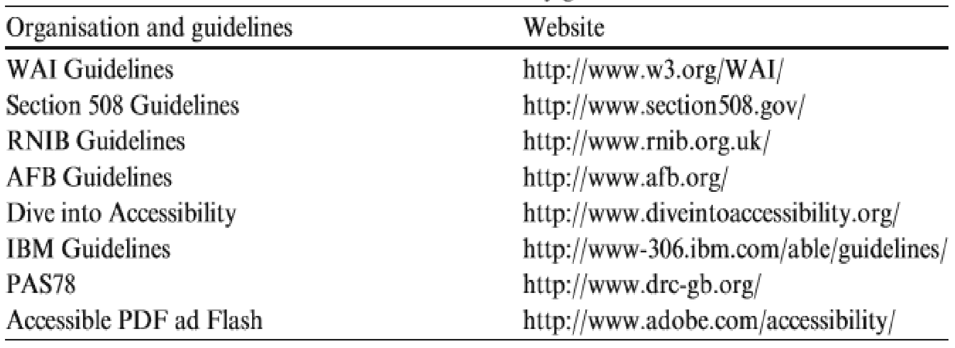
These tools are important for Web accessibility as they provide a medium for designers or authors to validate their pages against published guidelines without actually reading and manually applying them (Paciello 2000). Although there has been extensive work in the degree and development of these tools, automation is still limited (Yesilada et al. 2004). While it is likely that there are certain accessibility issues that cannot be fully automated (e.g., checking the quality of alternative text provided for images), these tools still provide incomplete automation and complex outputs. Similarly, there are also tools to transform Web pages into a more accessible form for disabled users (see Transcoding). Client-side rendering and transformation tools try to remould Web pages into user-centric presentations. This may either be in the form of a custom browser built to enhance the interaction of people with specific disabilities or as extensions to mainstream browsers or as proxies which modify Web documents as they are delivered to the user. However, most of these tools lack an understanding of disabled users' interaction with Web pages and their requirements. In order to address such user-requirement issues, some effort has been directed towards improving the tool support for designers (see Authoring Tools) building accessibility support in at the source.

**Web accessibility** refers to making websites usable for people of all types of abilities and disabilities, regardless of what browsing technology they are using. Since the web is an important resource of information for millions of people at all levels, accessible websites can help people with disabilities too to participate and contribute more actively in society. Accessible website help in making people with disabilities self reliant by providing facilities like online shopping, bill payments, online trading, online ticketing, online banking, etc. conveniently from their home. The physically impaired users will appreciate and return to websites that they can access easily, especially if the website is transaction based such as banking websites. Therefore, it is important to ensure that people with disabilities, have equal opportunities to benefit from the Web, especially from online public services. Therefore, accessibility of online content has become extremely important (Kaur, & Dani, 2014).

Although an accessible Web means unprecedented access to information for people with disabilities, recent research suggests that the best practice on accessibility has not yet been achieved. For example, Kelly (2002) found the accessibility of the high street stores, banks, and universities in the UK extremely disappointing. Eva (2002) surveyed 20 'Flagship' governmental Web sites in the UK and concluded that 75% needed immediate attention in one area or another. The Disability Rights Commission (DRC) conducted an extensive user evaluation, whose report (DRC) concludes that most Web sites (81%) fail to satisfy even basic accessibility requirements. The Web plays an increasingly important role in many areas (e.g., education, government), so an accessible Web that allows people with disabilities to actively participate in society is essential for equal opportunities in those areas. Furthermore, Web accessibility is not only a social issue but it is also becoming a legal requirement (Paciello 2000, Thatcher et al. 2002).

Nations and continents including the UK, Australia, Canada, and the United States are approving specific legislation to enforce Web accessibility. Web accessibility depends on several different components of Web development and interaction working together, including Web software (tools), Web developers (people) and content (e.g., type, size, complexity, etc.) (Chisholm and Henry 2005). The W3C Web Accessibility Initiative (WAI) 1 recognises these difficulties and provides guidelines for each of these interdependent components: (i) Authoring Tool Accessibility Guidelines (ATAG) which address software used to create Web sites (Treviranus et al. 2000); (ii) Web Content Accessibility Guidelines (WCAG) which address the information in a Web site, including text, images, forms, sounds, and so on (Chisholm et al. 1999); (iii) User Agent Accessibility Guidelines (UAAG) which address Web browsers and media players, and relate to assistive technologies (Gunderson and Jacobs 1999). There are also other organisations that provide accessibility guidelines such as shown in the table below:

**Table 1** Web accessibility guidelines



| Organisation and guidelines | Website |
|---|---|
| WAI Guidelines | http://www.w3.org/WAI/ |
| Section 508 Guidelines | http://www.section508.gov/ |
| RNIB Guidelines | http://www.rnib.org.uk/ |
| AFB Guidelines | http://www.afb.org/ |
| Dive into Accessibility | http://www.diveintoaccessibility.org/ |
| IBM Guidelines | http://www-306.ibm.com/able/guidelines/ |
| PAS78 | http://www.drc-gb.org/ |
| Accessible PDF ad Flash | http://www.adobe.com/accessibility/ |

W3C's Web Accessibility Initiative developed Web Content Accessibility Guidelines (WCAG 1.0) in 1999 with the aim of providing a single shared standard for web content accessibility that meets needs of individuals, organizations and governments internationally [15]. Later they were succeeded by WCAG 2.0 published in 2008 which are more comprehensive set of guidelines that are compatible with past, present and future technologies [16]. Refer to the following link for the WCAG 2.0: http://www.w3.org/TR/WCAG20/. Although it is possible to conform either to WCAG 1.0 or to WCAG 2.0 (or both), the W3C recommends that new and updated content use WCAG 2.0. The W3C also recommends that Web accessibility policies reference WCAG 2.0.

**What is WCAG 2.0 ? WCAG 2.0** covers a wide range of recommendations for making Web content more accessible. Following these guidelines will make content accessible to a wider range of people with disabilities, including blindness and low vision, deafness and hearing loss, learning disabilities, cognitive limitations, limited movement, speech disabilities, photosensitivity and combinations of these. Following these guidelines will also often make your Web content more usable to users in general.

**Assistive technologies that are important in the context of WCAG 2.0 include the following**:

i. **Screen magnifiers**, and other visual reading assistants, which are used by people with visual, perceptual and physical print disabilities to change text font, size, spacing, color, synchronization with speech, etc. in order to improve the visual readability of rendered text and images;
ii. **Screen readers**, which are used by people who are blind to read textual information through synthesized speech or braille;
iii. **Text-to-speech** software, which is used by some people with cognitive, language, and learning disabilities to convert text into synthetic speech;
iv. **Speech recognition** software, which may be used by people who have some physical disabilities;
v. **Alternative keyboards**, which are used by people with certain physical disabilities to simulate the keyboard (including alternate keyboards that use head pointers, single switches, sip/puff and other special input devices.).
vi. **Alternative pointing devices**, which are used by people with certain physical disabilities to simulate mouse pointing and button activations.

Summary of assistive technologies, disabilities and purpose

| S/N | Assistive technology | Disability (s) | Purpose |
|---|---|---|---|
| 1. | Screen magnifier | Visual, perceptual and physical print disabilities. | To change text font, size, spacing, color, synchronization with speech, etc. |
| 2. | Screen reader | Blindness | To read textual information through synthesized speech or braille. |
| 3. | Head pointer | People who have no use of their hands. | To push keys on the keyboard. |
| 4. | Text to Speech software | Cognitive, language, and learning disabilities. | To convert text into synthetic speech. |
| 5. | Speech input software | People with difficulty in typing. | To type text and control the computer including mouse actions. |

WCAG 2.0 is a W3C Recommendation published in December 2008. These guidelines are compatible with both backward and future technologies. WCAG 2.0 consists of four general principles with 12 guidelines that comprise of 61 success criteria which represent testable entity. The principles provide the foundation for web accessibility. These principles are:

| **Principle** | **Description** |
|---|---|
| **Perceivable** | Users must be able to perceive (see or notice) the information being presented (it must not be undetectable by all of their senses). |
| **Operable** | Users must be able to operate the interface (the interface cannot require interaction that a user cannot perform). |
| **Understandable** | Users must be able to understand the information as well as the operation of the user interface (the content or operation cannot be beyond their understanding). |
| **Robust** | Users must be able to access the content as technologies advance (the content should remain accessible as technologies and user agents evolve). |

## Web Usability

Usability goals are business goals. Web sites that are hard to use frustrate customers, forfeit revenue, and erode brands. Executives can apply a disciplined approach to improve all aspects of ease-of-use. Start with usability reviews to assess specific flaws and understand their causes. Then fix the right problems through action-driven design practices. Finally, maintain usability with changes in business processes.

The 5 key principles of good website usability are described as follows:

### 1. Availability and Accessibility

Let's start with a basic, yet central aspect of usability: the availability and accessibility of your site. If people try to access your website and it doesn't work — for whatever reason –your website becomes worthless. Here are a few of the basics of availability and accessibility,

**Server uptime** – It's important to ensure your visitors don't get an error trying to load your site. Invest in good hosting.
**Broken links** – Double check that there are no dead links on your site. Nothing sends a visitor back to Google search results faster than a 404 page.
**Mobile responsiveness** – Make sure your site can handle different screen sizes and slow connections.

## 2. Clarity

You could say the core of usability is clarity.
If you distract or confuse your visitors, they will either need more time to find what they came for, or they might forget their initial goal all together. Either way, they will not experience your website as user-friendly and chances are that they leave dissatisfied and with no intention of coming back. Visitors come to your site with certain goals in mind. It is your job to help them reach these goals as quickly as possible. If you can manage to do that, your visitors will be pleased and you have laid the groundwork for a positive experience.
A clear and usable design can be achieved through:
**Simplicity** – Focus on what's important. If you don't distract your visitors they will be more likely to do what you want them to do.
**Familiarity** – Stick to what people already know. There is nothing wrong with looking at other sites for inspiration.
**Consistency** – Don't get cute. Create a consistent experience across your entire website to keep your visitors mind at ease.
**Guidance** – Take your visitors by the hand. Don't expect your visitors to explore your site all on their own. Instead, guide them through your site and show them what you have to offer.
**Direct feedback** – Feedback is essential to any interaction. The moment people interact with your site, make sure to offer an indication of success or failure of their actions.
**Good information architecture** – Understand your visitors' mental models and how they would expect you to structure the content on your site.

## 3. Learnability

Learnability is another important aspect of usability. It should be your goal to design intuitive interfaces — interfaces that don't require instructions, or even a long process of trial and error to figure them out. Key to intuitive design is to make use of what people already know, or create something new that is easy to learn. By now, people are familiar with a lot of design concepts used on the web. By using these concepts consistently, you meet your visitors' expectations. This way, you help them reach their goals more quickly. As human beings, we like patterns and recognition, which is why we are better at handling familiar situations rather than unfamiliar ones. If you use new concepts in your design, make sure to use them consistently and give people a hand during the initial learning phase. For example, you can offer additional information, or instructions the first time they use your site or product. Keep it simple and visual to help people remember new concepts.

## 4. Credibility

Credibility is a crucial aspect of any website.
Even if people find the content they are looking for, if they don't trust you, that content is worthless. Your website could cause site visitors to be skeptical about your business in any number of ways including whether or not you really exist, your reputation, or the quality of your content. It is important that people know you are a real company with real people. Offer a clear "***About Us***" page together with your contact details and if possible a physical address.

Of course your content also plays an important role for the perceived trustworthiness of your site. Make sure you are honest and precise about your content. Avoid mistakes, such as incorrect grammar or misspellings. Don't be modest about your expertise. If you are an expert in your field, make sure people know it. For example, you can show third-party testimonials, work references, or the number of your social media followers to win your visitors over.

### 5. Relevancy

Last but not least, relevancy contributes to good website usability. It is not enough that your website is clear, your content must also be relevant. Again, it is essential that you know your users and why they visit your site. Start with defining who your users are. Second, talk to them to find out what their goals are when visiting your site. Third, define user scenarios that demonstrate in which situation people visit your site to find what kind of content. Any design decision that you make should result in a more user-friendly website for your users.

## Human centred software development and evaluation

This topic aims at bridging the gap between the field of software engineering (SE) and Human Computer Interaction (HCI), and addresses the concerns of integrating usability and user centered systems design into the development process. This can be done by defining techniques, tools and practices that can fit into the entire software engineering lifecycle as well as by defining ways of addressing the knowledge and skills needed, and the attitudes and basic values that a user centered development methodology requires. Usability tests and user-centered design techniques are now recognized as important mile stones in the development of interactive applications including GUI oriented applications, e-commerce web sites, mobile services and eventually wearable technology. However, the problems suffered by many application development projects suggest that this recognition has yet to be reflected into Software Engineering methods.

Developing interactive software is all about people: not only about the people that will use the software, but also about those who develop it. Most developers of interactive software deliver some sort of enhanced support for end-users, and as such, their knowledge about the users and the use situation is crucial to the outcome of the process. Although there are numerous methods and techniques to capture information about the users and tasks, it is the attitude and basic values of those who develop the software that will inevitably make a difference in the results.

Several studies have shown that 80% of total maintenance costs are related to user's problems with the system and not technical bugs (Boehm, 1991). Among them, 64% are usability problems (Landauer, 1995). A survey of over 8000 projects undertaken by 350 US companies revealed that one third of the projects were never completed and one half succeeded only partially, that is, with partial functionalities, major cost overruns, and significant delays (Standish Group, 1995). Executive managers identified the major source of such failures from poor requirements (about half of the responses) — more specifically, the lack of user involvement (13%), requirements incompleteness (12%), changing requirements (11%), unrealistic expectations (6%), and unclear objectives (5%).

For a more detailed discussion of cost-justifying usability efforts as a whole, independent of specific UE methods, consult for example Mayhew, 1999, Landauer, 1995, Karat, 1997, and Donahue, 2001,. These problems are mainly due to the fact that in developing highly interactive software with a significant user interface, most software engineering methodologies do not propose any mechanisms for: (1) explicitly and empirically identifying and specifying user needs and usability requirements, and (2) testing and validating requirements with end-users before, during, and after the development. As a consequence, the developed systems generally meet all functional requirements, and yet are difficult to use with effectiveness, efficiency and satisfaction. The lack of adequate methodologies explains a large part of the frequently observed phenomenon whereby large numbers of change requests to modify are made after its deployment. Human-centered design (HCD) philosophy and related usability engineering (UE) methods provide powerful solutions to such problems (Norman and Draper, 1986; Vredenburg, 2003; Mayhew, 1999). However, wide spread software engineering methods, such as RUP (Rational Unified Process) or the more recent agile development approaches, still lack explicit integration of HCI/UE methods and processes (Kazman et al., 2003; Seffah and Metzker, 2004).

Today, even if software development teams recognize its appropriateness and powerfulness, usability remains the province of visionaries, isolated departments, enlightened software practitioners and large organizations, rather than the everyday practice of the typical software developer. Knowledge and theory is still scarce about how to efficiently and smoothly incorporate UE methods into established software development processes. While standards such as ISO 13407 (Human-Centered Design Processes for Interactive Systems) provide a detailed description of the major UCD activities as well as strategies to assess an organization's capability to adopt HCD practices, they lack guidance on how to effectively integrate usability in a specific development team, project or context. Often, it remains unclear to software and UE professionals if, and why, certain UE tools and methods are better suited than others in a certain development context. Moreover, HCD has been historically presented as the opposite, and sometimes as a replacement, to the system-driven philosophy generally used in software engineering (Norman and Draper, 1986).

The reality is that UE and software engineering techniques each have their own strengths and weaknesses and their objectives overlap in some areas but differ in others. UE methods should be a core part of every software development activity, yet despite their well-documented paybacks, they remain to be widely adopted. We argue that an integrated framework that incorporates design, development and evaluation principles from both fields will bring more effective use of UE within software development. However, the empirical evidence required can be extracted indirectly by deploying UE methods in practice and studying their adoption by practitioners. In industrial software development projects, data on the perceived quality of UE techniques can help in understanding how to integrate UE on a case by case basis.

## Human Centred Design Activities

Figure 1 below shows the relationship among each Human Centred Design (HCD) activities. As shown in the figure 1 below, HCD has six activities as follows:

(1) Plan the human-centred design process which is a decision phase of a project to apply HCD to a target system.

(2) Understand and Specify the context of use which is a phase that a project get information how user uses a target system.

(3) Specify the user needs and the user requirements which is a phase to extract users' needs to a target system and to specify the needs.

(4) Produce design solution to meet user requirements which is a phase to make a prototype or production according to the specification.

(5) Evaluate the design against requirements which evaluates whether the design meets the requirements.

(6) Design solution meet user requirements which evaluates whether the solution meets requirements.

In these, 2), 3), 4) and 5) are main activities in HCD process. These activities and process are similar to development process. However, as shown in Fig. 2, these activities are applied to each phase in development process upper phase in development process is more important for developing product, system or service with high usability. HCD is a method to give better UX to stakeholders and to provide system and product with high usability for users and stakeholders [2].

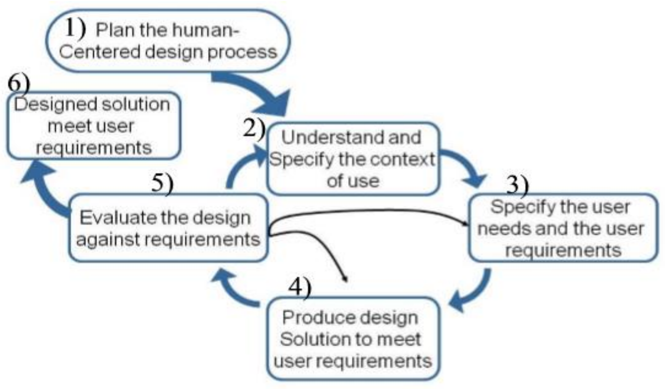

1) Plan the human-Centered design process
6) Designed solution meet user requirements
2) Understand and Specify the context of use
3) Specify the user needs and the user requirements
5) Evaluate the design against requirements
4) Produce design Solution to meet user requirements

**Figure 1**: Relationship among each human centred activities

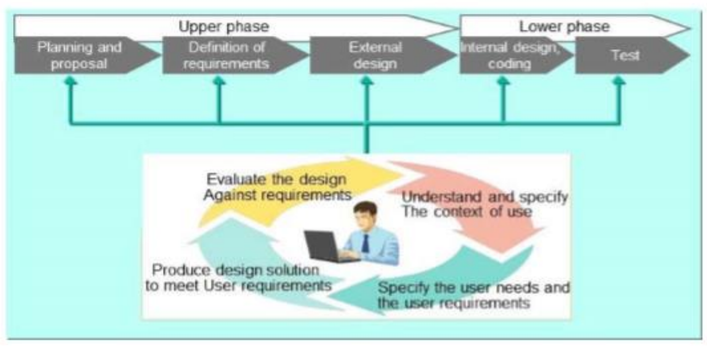

Upper phase
Lower phase
Planning and proposal
Definition of requirements
External design
Internal design, coding
Test
Evaluate the design Against requirements
Understand and specify The context of use
Produce design solution to meet User requirements
Specify the user needs and the user requirements

**Figure 2**: The relationship among human centred design activities and development process

**This lecture note is for students and teaching purpose only, the contents of the lecture note were adopted from:**

Benyon, D. (2014). Designing Interactive Systems: A comprehensive guide to HCI, UX and interaction design, 3/E.

Blomkvist, S. (2005). Towards a model for bridging agile development and user-centered design. In *Human-centered software engineering—integrating usability in the software development lifecycle* (pp. 219-244). Springer, Dordrecht.

Dix, A. (2016). Human–computer interaction, foundations and new paradigms. *Journal of Visual Languages & Computing*.

Fukuzumi, S. I., Noda, N., & Tanikawa, Y. (2017, May). How to apply human-centered design process (HCDP) to software development process?. In *Design and Innovation in Software Engineering (DISE), 2017 IEEE/ACM 1st International Workshop on* (pp. 13-16). IEEE.

Harper, S., & Chen, A. Q. (2012). Web accessibility guidelines. *World Wide Web*, *15*(1), 61-88.

Kaur, A., & Dani, D. (2014). Banking websites in India: an accessibility evaluation. *CSI transactions on ICT*, 2(1), 23-34.

Olsen, D. (2009). *Building interactive systems: principles for human-computer interaction*. Cengage Learning.

Pressman, R. S. (2005). *Software engineering: a practitioner's approach*. Palgrave Macmillan.

Pressman, R. S., & Lowe, D. (2000). Web Engineering. *Software Engineering: A Practitioner's Perspective*, 769-798.

https://www.interaction-design.org/literature/topics/voice-user-interfaces.

Sabina Idler https://www.crazyegg.com/blog/principles-website-usability/

Seffah, A., Gulliksen, J., & Desmarais, M. (2005). An introduction to human-centered software engineering: Integrating usability in the development process.

Viikki, K., & Palviainen, J. (2011, August). Integrating Human-Centered Design into Software Development: An Action Research Study in the Automation Industry. In *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on* (pp. 313-320). IEEE.