32.1 Introduction

The essence of usability as a software development goal is that tools and systems can be designed to facilitate and enhance desirable experiences for their users—ease, transparency, engagement, fun, usefulness, self-efficacy, satisfaction—as well as to mitigate undesirable experiences, such as confusion, frustration, anger, disappointment, lack of confidence, and so forth. Usability is a nonfunctional requirement in that it describes the criteria for assessing the quality of a system beyond merely specifying the functions that the system performs, its functional requirements.

As a supporting framework for achieving usable systems, the insights, guiding principles, and methods of usability engineering are deceptively simple and straightforward. The first is that ensuring the usability of interactive systems is necessarily a process: usability cannot be reduced to fixed laws, scripts, or guidelines. The usability process must be initiated at the beginning of the system development process, and continue throughout and beyond the traditional scope of system development. This is a painfully important insight, because it is far more difficult and costly to manage a process than to follow a rule.

The second principle is that usability can indeed be engineered: to a considerable extent, usability can be systematically managed through replicable and teachable methods to meet explicit and measurable objectives (often called usability requirements). Usability can be empirically tracked and iteratively improved throughout the system development process to ensure better outcomes. In other words, usability is not a matter of trial and error in development, followed by empirical checking at the end to see if things worked out.

The third principle is that usability is an open concept: conceptions of usability have evolved throughout the past 40 years, and continue to evolve. One obvious reason for this is that the nature of interactive systems has evolved very rapidly. The concepts appropriate to understand and manage the usability of a programming environment in 1977, a PC database in 1987, a website in 1997, or a mobile personal system in 2007 are all quite distinct. But, another reason for the continuing evolution of usability is
that our understanding of usability and our arsenal of usability engineering tools and methods have improved continually. The trajectory of usability through the past four decades is that it has become far more articulated, far more nuanced, and far more comprehensive.

These three principles are obviously interdependent and contingent. Thus, the scope and rate of change in interactive systems is a key factor in the dynamic nature of usability itself, which, in turn, is part of why usability must be managed as a process. It is difficult to see any narrowing or slowing in the evolution of interactive technologies and applications. Change is clearly accelerating.

### 32.2 Emergence of Usability Engineering

#### 32.2.1 Software Psychology

The concept of usability and the practices of usability engineering began to take shape in the context of 1970s' mainframe/terminals system configurations with line-oriented displays, batch processing for many interactions, and purely textual command-driven interfaces (sometimes with cosmetic menus). Early usability research struggled against two persistent fallacies of system development. The first is the tendency of programmers and designers to assume that their own experiences are valid indicators of what other people will experience in using their software. This seems naively optimistic when stated in the abstract, but it is surprisingly seductive. One still sees it regularly among students in introductory software design classes, but in the 1970s, it was a pervasive attitude among software professionals. The second fallacy is that usability flaws can usually be addressed through adding help and training support to the overall system. In fact, serious usability issues cannot be addressed by help and training. Indeed, designers must always recognize that adding help and training is ipso facto adding complexity to the overall system design.

Throughout the 1970s, usability was investigated as part of a project called “software psychology” (Shneiderman, 1980). This effort tried to confront the fallacies of usability with scientific studies of usability phenomena. Of course, in the 1970s, ordinary people did not use computers, and there were no “users” in the sense we understand that term now. Rather, there were a variety of data processing professionals who designed, implemented, maintained, and used mainframe systems and applications, for the most part, as their full-time jobs. The vision of software psychology was to apply concepts and methods of experimental psychology to better understand the problems of professional programmers, including the then-emerging ranks of application programmers. Three examples of early software psychology contributions are with respect to commands and command languages, design problem solving, and collaborative work; these will be discussed in the following paragraphs.

There are many fairly direct mappings of phenomena from experimental psychology into programming. Thus, especially in the 1970s, programs were long lists of expressions, and command interfaces were long lists of command-function pairs. Psychologists had demonstrated severe limitations of human working memory, including distinctive patterns such as primacy and recency, the tendency in memorizing lists to recall items early and late in the list better than items in the middle of the list. It was also known that people spontaneously impose sophisticated structuring on verbal material. Thus, if a command “down” moved the cursor ahead one record, people automatically assume there will be a command “up” and that it will move the pointer back one record. From this, it could be expected that memory failures, including particular patterns such as primacy and recency, and confusions created by badly designed command languages would be pervasive and debilitating in a text-based computing paradigm.

And, indeed they were (e.g., Card et al., 1983; Carroll, 1985; Grudin and Barnard, 1984; Norman, 1988).

The psychology of problem solving had demonstrated that humans often decompose complex problems into subproblems and lose track of the dependencies across a subproblem, that they rely on metaphors and analogies in simplifying complex problems, that they frequently underanalyze problems, that they mistake merely familiar problem elements as being more important elements, and that they rarely optimize solutions (Carroll, 1987; Kahneman and Tversky, 1979; Reitman, 1965; Simon, 1972).
Of course, people are also still the best problem-solving engines we know of, and regularly generate deeply creative insights into complex problems, often “solving” problems that were too complex to even be stated precisely. Technologies can be crafted to address the inherent flaws of human problem solving, for example, representing problem analyses and decompositions in ways that help to expose missing parts (Norman, 1988). Technologies can also strengthen the strengths of human problem solving, for example, suggesting criteria for evaluating metaphors and analogies (Carroll and Kellogg, 1989). The technologies of the 1970s were poorly matched to the characteristics of human problem solvers, and this became more obvious as the decade progressed.

Finally, the psychology of programming identified issues and challenges in the coordination of work in teams. Among the most ambitious software projects of the 1960s was the development of IBM’s Operating System 360 (Brooks, 1975). The size and schedule of this project required a large team, and a standard conception of the time was the “man-month,” in this case, the work a single programmer could contribute in a month. However, reflections on the project revealed that adding team members did not increase team productivity linearly. Indeed, there was a severe drop in productivity owing to the need for additional team members to spend much more time coordinating their individual efforts. In this case, the analysis of software development experience slightly led the basic psychology of collective effort that is its foundation. But since the 1960s, social psychologists have extensively investigated the issues in optimizing collective efforts (West, 2012).

32.2.2 Early Human–Computer Interaction

Software psychology primarily focused on characterizing usability issues with respect to a scientific foundation of psychological concepts and methods. It was largely descriptive and reflective. In the early 1980s, computing was broadly transformed by the emergence of the personal computer. In the course of only a few years, the problems of professional programmers receded as the focus of research on usability. Software psychology was absorbed into a larger area called Human–Computer Interaction (HCI). HCI expanded the software psychology agenda to be more proactive with respect to new concepts, tools, and methods, including a fundamental rethinking of the software development process. This was good timing; the “waterfall” conceptions of software development had failed even for 1970’s conceptions of users and applications. In the 1980s, developers were eagerly seeking new paradigms that could help them design for the growing PC world, where the focus was the user interface and the user. HCI took on this challenge, and through the next decade transformed computer science, establishing what is now taken for granted: software that is not highly usable is just a failed software.

Early HCI embraced three distinctive goals. All three contrast with the goals of software psychology, and all three positioned HCI as a focal area of challenge and change in computing. First, HCI was conceived of as an area in which new science and theory would emerge and develop, not merely as an application area for existing basic knowledge. Initially, HCI was an area of cognitive science, and later it also sought to make fundamental contributions to sociology and anthropology. Second, HCI was conceived of as an area in which new models and techniques for software design and development would emerge and develop, not merely as a project to enrich or improve existing software development models. Third, HCI was conceived of as a technology area in which new user interface software and new applications, new software architectures and tools, and even entirely new types of software would be developed and investigated. Remarkably, all of these goals were achieved in the next couple of decades.

In the area of science and theory, HCI adapted descriptive models from many realms of cognitive and social science, including visual and auditory perception, models of routine cognitive skill, mental models, common ground, distributed cognition, activity theory, social psychology, and sociological studies of work and of community (Carroll, 2003). HCI also was a primary incubator for new theoretical perspectives and technical achievements that have influenced basic sciences, including integrated information processing models (Card et al., 1983), analyses of situated actions (Suchman, 1987), and conceptions of distributed cognition (Hollan et al., 2000), activity (Kaptelinin and Nardi, 2012), and experience.
Human–Computer Interaction and User Experience (McCarthy and Wright, 2004). This incredibly varied science base is used in many ways in usability engineering, including design techniques (such as metaphorical design), design and evaluation tools (Bellamy et al., 2011), theoretically grounded design rationales (Carroll and Rosson, 2003), and conceptual frameworks for critiquing designs (McCarthy and Wright, 2004).

In the area of models and techniques for software development, HCI framed the principle of iterative development, the claim that systems can never be successfully specified without extensive prototyping and refinement (Brooks, 1975; Carroll and Rosson, 1985; Gould and Lewis, 1985). The principle of iterative development entrained several further paradigmatic ideas that have shaped the development process for interactive systems. The paradigm of user-centered design requires that system design goals be aligned with the goals of the eventual users of the system. This is sometimes achieved through empirical requirements development, empirical studies of user activities and preferences that are then used to guide envisionment of future systems. Sometimes, it is achieved through direct user participation in the design team, called participatory design.

Iterative, user centered, and participatory design alter the system development process in fundamental ways. These paradigmatic commitments entailed a wide variety of specific usability engineering concepts and techniques to gather data from or about users and to represent and interpret that data. For example, creating scenario descriptions of user activity—essentially stories—is now a core technique in interactive system design, often called scenario-based design (Carroll, 2000; Rosson and Carroll, 2002). As late as in the 1980s, this was seen as a radical approach to software development; today, not doing it would be radical.

In the area of software technology, HCI research produced software architectures and tools that modularized graphical user interfaces and their many successors among personal devices, embedded systems, immersive systems, etc. The initial design strategy was to isolate user interface functionality in a User Interface Management System (UIMS), to make it easier to adjust characteristics of the user interface through the course of design iterations, without entailing disruption to underlying system functionality. Although such a clean separation proved to be intractable, UIMS work helped to establish a software requirement of malleability to enable iterative design.

### 32.2.3 Managing Experience Design

HCI is inescapably a design discipline, but the term design has a variety of different meanings. HCI is a design in the sense that creating interactive systems is always an ill-structured problem; a problem that can only be stated precisely after it is solved (Reitman, 1965). But, interactive systems and applications have expanded into every arena of human activity, and as the richness of user experiences afforded and evoked by interactive systems has expanded, HCI has also become design in the sense that it shapes our interaction with the social and material world, such as consumer product designs, mass media designs, and so forth. Experience design, as it is sometimes called, is now an important focus in HCI and usability engineering (Diller et al., 2005).

This entails a further expansion of what usability and usability engineering are about. The user’s experience is no longer just an issue of understanding what is going on well enough to achieve a specific task goal. That conception of usability is still valid, still necessary, but it is no longer sufficient for understanding the user experience. It can be argued that the user experience was always much more complicated than merely easy-to-learn/easy-to-use. As early as the mid-1980s, it was argued that usable systems should be fun to interact with (Carroll and Thomas, 1988). One way to see this is that until the field had developed far enough to effectively and broadly address the simplest conceptions of usability, such as ease of learning and use, the much richer issues that are currently in focus could not get to center stage.

Throughout the past four decades, the concepts and practices of usability engineering have expanded to manage increasingly richer and more nuanced conceptions of usability. The field has been shaped by external factors, a technology context of new devices and networking infrastructures,
new systems, services, and applications, new user interface designs, and new types of users with an ever-widening range of interests and motivations, domain knowledge and knowledge of computing, etc. But it has also been shaped by an astounding diversity of skills and sensitivities among its researchers and practitioners. There is now a solid core of usability engineering tools and practices, as well as an active and rapidly advancing frontier of new concepts and techniques growing year by year.

### 32.3 Core Methods of Usability Engineering

The tools and practices of usability engineering are often applied within a life cycle framework, for instance, scenario-based usability engineering (Rosson and Carroll, 2002), contextual development (Beyer and Holtzblatt, 1997), or the wheel model (Hartson and Pyla, 2012). The goal of such frameworks is to recommend and demonstrate the use of a variety of techniques at different points, often in a repeated and iterative fashion, through the development of a system.

A high-level summary of how core usability engineering activities might be integrated within a scenario-based development (SBD) life cycle appears in Table 32.1 (the hypothetical example concerns a new e-commerce website). In the SBD framework, scenarios and associated design rationale serve as a central representation that assures a pervasive focus on the task goals and experiences a design team envisions for users. The scenarios also help team members with differing expertise (e.g., social vs. computational science) to establish and maintain common ground as they discuss usability engineering trade-offs. At different points through the cycle, the usability engineering team may use a range of methods, and in some cases the same basic method may be used in support of different usability development goals.

#### 32.3.1 Direct Observation

The most straightforward way to learn about and respond to human needs and preferences is to observe them as they carry out activities that rely on the system or design concept under development. The types of methods and analyses selected for these activities depend on many factors, including the goals of the observational studies, the resources available to the evaluation team, and the nature of the interactive system being developed. Depending on where a team is in the product development life cycle,

---

**Table 32.1** Usability Engineering Methods for Phases of Scenario-Based Design

<table>
<thead>
<tr>
<th>Life Cycle Phase</th>
<th>Examples of Usability Engineering Methods</th>
</tr>
</thead>
</table>
| **Problem analysis**: Scenarios synthesizing current practices with associated trade-offs | • Contextual inquiry of administrative employees using BuyOnlineNow.com to order office supplies  
• Convenience survey of current users of amazon.com  
• Focus groups discussing personas and scenarios synthesized from contextual inquiry and survey |
| **Design**: Increasingly detailed scenarios of new activities with associated rationale | • Claims analysis and reasoning about trade-offs in problem scenarios to guide redesign  
• Participatory design sessions with administrative employees mocking up rough design scenarios for online purchasing tasks |
| **Evaluation**: Formative evaluation that guides redesign, complemented by summative evaluation to measure success | • Think-aloud study of administrative employees working with a low-fidelity prototype  
• Performance study measuring time and errors for several benchmark tasks (e.g., locating item, completing purchase)  
• Convenience survey of users of a new system once fielded to gather subjective reactions, loyalty, etc. |
there may be different evaluation goals in focus (Scriven, 1967). For example, early in the life cycle, *formative* methods are an important mechanism for producing rich and often qualitative findings that can guide iterative design activities. Toward the end of the life cycle, or at critical milestones, the team may instead use *summative* methods to characterize whether and how well usability objectives have been met. In this sense, some of the very early observational studies may include analysis of precursor systems that the target users already employ for similar tasks.

A usability evaluation team may have the luxury of a relatively open-ended mission, spanning many months and having access to a diverse team of usability experts and target users, but more commonly their activities are constrained by time and other resources available. Because of such constraints, the team may need to address their studies to a small set of features or usability questions that imply the use of specific observational methods. For instance, answering a question about two different dialog designs for an e-commerce system implies a performance-oriented observation, perhaps complemented by user satisfaction. In contrast, a more exploratory study of a novel information structure for the same e-commerce site might be best served by a study gathering users’ verbal protocols. In the following, we summarize several common observational methods found in usability engineering, ordered roughly by the goals and resources available to the team.

### 32.3.1.1 Workplace Ethnography

A workplace ethnography would typically be carried out either in the early phases of a project, or perhaps after a project has been completed and fielded. Its goals are to study the established use of some system(s) of interest, without any intervention or structure provided by the observers. In fact, the goal of an ethnographer is to “disappear” into the background of a workplace setting. The primary source of data is the analyst’s field notes about what he or she is observing; evaluators may introduce more comprehensive records like audio- or video-recordings, but generally would do so only if these recordings can be made in an unobtrusive fashion. In a pure ethnography, the evaluator would attempt to record everything that is happening, perhaps with a special focus on technology usage episode, but to do this in as unbiased as fashion as possible. Because such evaluations rely entirely on natural work settings and activity rhythms to reveal user tasks or reactions “of interest,” they generally require long time periods—weeks or even months—of regular observation. A classic example of workplace ethnography that was used to fuel design thinking can be found in the study of Hughes et al. (1992) of an air traffic control room.

The ethnographic evaluation that is more commonly found in usability engineering contexts is more brief and necessarily more focused in its observation, sometimes referred to as “quick and dirty” or rapid ethnography (Hughes et al., 1995). The general concepts of rich observation and analyst-centered interpretation are still in place, but the abbreviated length and reduced scope assume that an expert observer can enter a situation where design-relevant behavior is underway, and in a relatively short time extract key practices, preferences, and obstacles from workers’ behaviors. In the e-commerce example, a quick and dirty ethnographic study may involve observations of shoppers in a store as well as interviews with people who have used existing e-commerce systems; it might also involve the collection and analysis of web interaction logs and context-specific artifacts such as a “filled” online shopping cart, a shopper profile, recommendations made by the system, and so on.

### 32.3.1.2 Contextual Inquiry

Contextual inquiry provides a more focused approach to field observation; these methods were first described and implemented by Hugh Beyers and Karen Holtzblatt (Beyer and Holtzblatt, 1995) as a centerpiece of their contextual design framework. Like ethnographic methods, this observational technique requires evaluators to collect data from the field as workers engage in their everyday activities. It is particularly useful during the requirements analysis conducted in the early phases of interactive product design, because it produces a number of analysis documents that encode important artifacts currently in use, including technology, physical, social, or organizational factors that impinge on current
practices, and so on. The goals of this technique are similar to rapid ethnography, but the data collection includes more intervention from the observer.

A typical setting for contextual inquiry might include an observer who sits near the workstation or desk of a worker engaged in everyday tasks. The observer watches and takes notes (perhaps also digital recordings), as would an ethnographer. However, the observer’s presence is more salient to the worker (e.g., because of proximity) and she is expected to ask questions at times when something of “interest” takes place. This requires the observer to begin the session with a particular focus, perhaps even a set of task-related questions, and to inject these concerns when as relevant. Early on in a system design process, the data collection may be guided by very generic prompts, for example, “Why did you do that?” or “What just happened?” but at times it may be appropriate to make these more specific, for instance, “How did you know it was time to press the Submit button just now?” In our e-commerce example, a contextual inquiry could be directed at either (or both) shoppers in a store setting (e.g., shadowing and observing but also asking shoppers about their search and decision making) or shoppers who interact with an existing e-commerce site.

32.3.1.3 Think Aloud Observations

When usability evaluators want to probe the goals, expectations, and reactions of a user to a particular set of user interface features or controls, they may conduct a session in which they ask users to “think out loud” about their experiences while attempting a task. Such sessions could be done in a field setting (e.g., in conjunction with contextual inquiry visits), but more commonly take place in a laboratory setting where participants are recruited to work with a system while their mental experiences are vocalized and recorded. The resulting auditory file is typically synchronized with a log of their system interactions, and is often termed a verbal protocol (e.g., as in the methods pioneered by cognitive psychologists studying problem solving [Ericsson and Simon, 1980]).

In a typical think aloud session, a representative user will be asked to attempt a specified task (e.g., locate and order *The Girl with the Dragon Tattoo* using Amazon.com), while also saying out loud her moment-to-moment interpretations of what she sees, her plans for what to do next, and her reactions to the system’s responses at each step. The evaluator’s goal in this is to infer important aspects of the user’s mental model (Carroll et al., 1987; Payne, 2003), that is, her current knowledge of what an interactive system can do for her and how to instruct it to meet these goals. These results are particularly useful if the software being designed relies on novel or unusual information abstractions (e.g., new vocabulary or conceptual relationships), or if it relies on user interface controls that have not been seen or used in similar systems. These studies are most useful when the primary objective for the system is that it requires little, if any, learning for new users. Because a think-aloud requirement creates a secondary—and often distracting—task with respect to the primary usage task, these methods would normally be used only to serve formative evaluation goals.

32.3.1.4 Human Performance Studies

The most common form of observational study—and what most usability practitioners mean when they talk about doing a user test—is a controlled study of representative users’ performance with an interactive system. The method is straightforward: a set of typical tasks is designed to exercise the aspects of a system needing feedback, and users with characteristics similar to those of the project’s target users are recruited to carry out these tasks (Card et al., 1983).

For instance, continuing with the e-commerce example, a team might design four to five product search or purchasing tasks that involve different subdialogs within a high-fidelity prototype; perhaps some rely on recommendations or reviews, whereas others are more specific examples of item location or purchase decisions. The data collected in these cases would be objective in form and might include: (1) the number of steps required to complete the task; (2) the actual path followed; (3) the number of mistakes taken; (4) the time required to complete the task, and so on. These data would be summarized for comparison to preestablished task performance expectations, perhaps based on competitive analysis,
on pilot studies with an earlier version of this system, or mathematical human performance models. In a formative evaluation setting, such data would likely be complemented by other methods such as those described earlier, but for summative evaluation, they might be used on their own to determine relative success or failure of the system with respect to user performance.

32.3.2 User Expectations and Preferences

Beyond direct observation of behavior, usability engineers normally seek to gain insight into users’ expectations, beliefs, or experiences with respect to the system or design concept. One challenge in this process is assuring that test users report their actual feelings as accurately as they are able. There is a general tendency to exhibit socially desirable behavior, and in a system test, this may bias participants to be more positive than their experiences warrant. Another challenge is to evoke reactions that are nuanced and specific, rather than “Sure, I like this!” or “There is no way I would ever use this system!” Finally, usability engineers must be careful to frame users’ reports of their expectations and preferences as subjective data that can be affected by a wide variety of individual differences and contextual factors.

32.3.2.1 Focus Groups

Usability engineers may conduct focus group sessions in the early phases of a software development project; group discussions are particularly useful for exploratory data collection. A group of representative users (or perhaps other stakeholders, for instance, decision makers) is gathered together and guided through a discussion about issues relevant to the current phase of system development. Depending on the level of design and development, the discussion may be anchored by demonstration of a design storyboard or even an early prototype. In SBD, scenarios are often communicated informally as textual narratives that revolve around personas (Cooper, 1997), but with a bit more effort, they can be elaborated using sketches or screen shots that capture key aspects of the imagined interaction sequence.

Early on in design, before system functionality or user interactions have been proposed, a focus group may be shown other existing technologies that are related to the design ideas being considered; in our e-commerce example, a session may start out with a few videotapes of people using amazon.com, so as to ground the discussion on particular aspects of current approaches. Depending on the time and resources available, the facilitator may rely instead on a semistructured script of questions or topics. Focus group discussions would normally be recorded for later review; a video of the participants interacting with each other can also be helpful in later analysis of when and how different ideas were generated. A typical result from a focus group would be a summary of ideas raised, perhaps organized by the question or topic that evoked the ideas. If the group has discussed specific design features, there may be a more detailed analysis of the positive and negative comments that were made, or the suggestions for changes.

32.3.2.2 Interviews

An interview is similar to a focus group, but the questions are directed to an individual. As with focus groups, the interview may be more or less structured and may include demonstrations. Typically, the interview is semistructured such that it includes a set of high-level questions but the evaluator is left free to pursue interesting details that come up, or even to engage in an entirely new topic that seems to be a rich source of information from a given participant.

In an interview, a person answers questions individually, so it is important to establish rapport with her quickly so that she feels comfortable enough to expand on her personal experiences and attitudes. One technique for establishing rapport is to begin the questioning with some background discussion about the participant’s current work (or leisure, education, etc.) setting. General background information is simple and easy to provide, and has the added advantage of providing participant-specific context that the interviewer can later inject into questions or follow-up probes. For example, in the e-commerce setting, the usability engineers might ask questions about how often and for what sorts of purposes shopping trips are made, as well as what the interviewee most enjoys or finds irritating about shopping.
as an activity. Another useful technique when interviewing is to prompt for examples or illustrations of points being made; by asking the participant to remember and describe specific cases, the interviewer can both clarify what a participant is trying to convey and, in general, evoke a richer experiential report.

### 32.3.2.3 Questionnaires and Surveys

Usability engineers often develop questionnaires or surveys to gather users’ expectations or reactions as a complement to open-ended methods. The analysis of qualitative data gathered during direct observation (e.g., in a think aloud study) can be tedious and time-consuming. In contrast, a well-designed survey can gather probe a number of specific user experiences using rating scales or other item types that are simple for users to complete and straightforward to summarize. In fact, usability professionals have adapted the psychometric methods of sociology and psychology to generate and diseminate several validated scales for usability assessment. For example, the QUIS instrument (Questionnaire for User Interface Satisfaction) was developed in the late 1980s as a research project in the HCI lab at the University of Maryland (Chin et al., 1988); QUIS has been validated and refined over the years and is available for licensing to practicing usability engineers (http://lap.umd.edu/quis).

Existing and validated scales are useful in positioning users’ general reactions or preferences. However, by their nature, such instruments are conceptualized at a relatively general level. QUIS includes subscales for general usability objectives such as screen design, vocabulary use, and support for learning. However, usability engineers who need to evaluate novel design features (e.g., a small touch screen or natural language input) will need to extend such instruments with items specifically oriented to assess these features. These additional items will be particularly critical in the formative phases of system development, when the design team is deciding which new ideas to pursue and how to refine them.

When used in tandem with direct observation studies, usability engineers typically prepare a brief background survey that is administered before the tasks are attempted; this survey can be used to gather participants’ expectations or other preexisting knowledge that may (1) influence how they approach the task; or may (2) change as a result of working with the new system. A posttask survey may repeat some of these items, enabling an analysis of changes in users’ expectations or preferences, contrasting their responses before and after their usage experience. For example, in the hypothetical e-commerce project that includes a novel interface for finding and reading product reviews, the background survey might ask users about the perceived cost of finding or absorbing product reviews, and compare ratings on this question to the same item presented after the users try out the new approach.

In addition to the development or adaptation of usability-related rating scales, usability questionnaires and surveys often include open-ended questions (e.g., describe the three things you liked the most about the system you just used). As for other qualitative methods, these items are particularly useful early on in the development process when the goal is to develop as rich an understanding as possible of users’ experiences. The open-ended items may also be used to probe a participant’s baseline comprehension of specific concepts or procedures that are incorporated into the system design. Responses to open-ended questions may be summarized or coded in a fashion similar to transcripts of interviews or focus groups.

Survey methods are also used for requirements gathering in advance of designing a system, or for field-testing of new software after it has been launched. In these cases, considerable effort should be spent in developing and refining a set of questions that addresses the trade-off between information gathering and survey length, simply because in general participants may not be willing to complete a survey that requires more than 10–15 min of their time.

### 32.3.2.4 Participatory Design

Focus groups, interviews, and surveys can be used to gather preferences or reactions to an existing or proposed interactive system. In these cases, the design team has the responsibility to make sense of the information gathered from users, determining whether and how best to refine the team’s current concepts or prototype system. A more direct way of injecting users’ preferences or concerns into a design project is to engage them as participants on the design team. Participatory design activities can take
place at specific points in the life cycle (Muller, 2007) or as a long-term evolutionary process (Carroll et al., 2000; Greenbaum and Kyng, 1991). The general concept is similar to a focus group; however, the goal is not to explore concerns and ideas but rather to generate new design concepts, often represented physically using mock-ups of varying levels of fidelity.

In projects that incorporate participatory design, a single set of representative users often meets repeatedly with the design team and collaborates directly on extending or refining the existing analysis or design representations. For example, early on the process they may help to analyze data collected from field studies, such as snippets from videos collected from earlier fieldwork (Chin et al., 1997). Once a problem situation has been analyzed, the group of prospective users may work with the designers to develop or refine user descriptions or task scenarios that envision new or revised activities (Carroll et al., 2000). As the design ideas become more concrete, the users may also help to sketch out user interface displays or interaction sequences (Muller, 1991). Finally, if the software is being constructed in an open and iterative fashion, and if appropriate end user development tools are provided, the user representatives may even operate directly on the software under construction to make changes to a prototype (Costabile et al., 2004).

32.3.3 Analytic Methods

Software projects do not always have the time or resources to recruit and involve representative users or their surrogates in the development process. Alternatively, they may be working within a task domain where detailed performance studies and optimizations are crucial to the success of the project (e.g., safety- or time-critical problem domains). In these situations, a usability engineer may choose to apply analytic methods—techniques that rely on the usability expertise of professionals to identify and manage usability concerns. Like empirical methods, analytic approaches vary in their precision and the corresponding cost of application.

32.3.3.1 Heuristic Evaluation

One class of analytic methods draws from heuristics or high-level guidelines to structure an analyst’s review of a design. The interactive software concepts under review might be preliminary (e.g., initial ideas that have been documented through scenarios or storyboards), or it may be a functioning prototype. The heuristics may be a list of best practices that have been summarized across a number of different projects—perhaps within a “family” of interactive applications such as the user interface guidelines used by Apple or Microsoft. They might also be drawn from expert usability practitioners (e.g., http://www.useit.com/papers/heuristic/heuristic_list.html) or HCI researchers (e.g., the guidelines found in Shneiderman et al., 2009). For instance, one common heuristic is to “Speak the user’s language,” implying a careful focus on the terms and phrasing used by the system’s user interface controls and other explanatory text. An analytic evaluation that used this guideline as a heuristic would result in a detailed critique of vocabulary-based issues, perhaps broken down into different segments of the user population. Of course, most such evaluations would be addressing multiple guidelines or heuristics at once, leading to an extensive report at the end of the analysis. Large companies often maintain internal guidelines that address usability issues; depending on the market for the product, one or more national or international standards may also be consulted (see e.g., http://zing.ncsl.nist.gov/iusr/documents/CISU-R-1R7432.pdf). Thus, a professional usability engineer is expected to gain expertise with many forms of standards and guidelines.

The process of applying usability heuristics and guidelines may range from an ongoing background activity (e.g., an e-commerce usability engineer would always be considering issues of web page accessibility by vision-impaired users) to concrete usability inspection episodes. For instance, an expert might conduct a usability inspection of the e-commerce application in our example by receiving a specified set of tasks for product browsing, searching, purchasing, and so on from the design team, and then enacting a careful step-by-step inspection of these tasks in light of the usability heuristics or guidelines provided.
by the organization. The inspection might be conducted by an individual expert, by a pair of experts, or by multiple experts conducting inspections independently. The general goal is to provide a report of usability problems that are evoked by the heuristics; the problems are often further classified by severity, the price of addressing the problem, or other company priorities.

### 32.3.3.2 Design Rationale

The analysis of a designed artifact’s design rationale draws from the precepts of ecological science originally pioneered by Gibson (1966, 1979) and Brunswick (1956). Brunswick showed how some of the central perceptual phenomena identified and investigated with line drawings, such as visual illusions, could be explained in terms of the ways that objects align and occlude in the physical world. Gibson showed that perception in movement permits direct recognition of higher-order optical properties such as the expansion pattern one sees in approaching a textured scene or surface. Gibson concluded that real-world perception typically consists of direct recognition of complex properties, which he called “affordances.” This contradicted the most fundamental assumption of perceptual psychology, namely, that higher-order properties, such as being an object or being something that can be poured, are inferred on the basis of piecemeal recognition of local features such as contours, edges, and angles. In HCI, the use of design rationale leverages affordances, arguing that a useful analysis for usability is to consider in detail the affordances—or consequences—of an artifact’s features for its use in the world. Typically, such analyses produce trade-off descriptions, because almost every system feature, one can imagine, implies a mix of usability pros and cons for different users in different settings.

One approach to developing and maintaining usability design rationale is claims analysis, a method used to hypothesize features of an artifact in use (e.g., as illustrated by a usage scenario) that have causal relations with a set of positive and negative consequences for end users (Carroll, 2000; Rosson and Carroll, 2002). Claims analysis was developed to articulate the causal relations inhering in artifacts-as-theory (Carroll and Kellogg, 1989). It has been incorporated as a central reasoning and documentation method in scenario-based design: throughout the evolution of the design scenarios, the design team reasons about the claims, attempting to envision new features that can address the negative consequences hypothesized, while also maintaining or enhancing the positives (Rosson and Carroll, 2002).

Claims development is a form of analytic evaluation. At the same time, the process can set up a structure for claims-based empirical evaluations—this becomes a form of mediated usability evaluation that synthesizes analytic and empirical evaluation methods (Carroll and Rosson, 1995; Scriven, 1967). The claims that are created through an analytic process embody hypotheses about what users will expect, attempt, or experience when engaged in a particular activity context (scenario). Such hypotheses can be tested specifically (e.g., in a controlled study that engages users in the relevant activity) or indirectly (e.g., through a generalization process that leads to broader hypotheses that are evaluated through a series of tests).

In the context of claims analysis, design artifacts under construction (assuming some context of use) always have one or more design features in focus; the analysis helps a design team to reflect on and learn from their design discussions and experimentation. Examples of claims analysis as a usability engineering method can be found in case studies of collaborative learning software (Chin, 2004), community networking software and activities (Carroll and Rosson, 2003; 2008), and developmental online communities (Rosson and Carroll, 2013).

### 32.3.3.3 Cognitive and Computational Models

Another class of analytic methods has its foundations in mathematical models of human information processing (Card et al., 1983). Rather than inspecting a piece of software to identify problematic features, the cognitive modeling approach requires one to build a formal model of the knowledge and procedures needed to complete a task. For example, in the GOMS method, these models require the decomposition of a user interaction task into goals, operators, methods, and selection rules. The cognitive model produced in this fashion can subsequently be translated into a computational model;
performance parameters can be estimated for the model’s primitive operations (e.g., pressing a key, moving a mouse, recognizing a word), and the resulting program can be executed to produce performance estimates (Hornoff and Halverson, 2003). Once a model has been created, it can be revised and re-executed to analyze whether specific changes in task design may lead to performance gains. The use of analytic methods at this level of detail is time-consuming, but may be cost-effective when the tasks being optimized are routine behaviors that will be executed hundreds of times per day by thousands of users (Gray et al., 1993; John and Kieras, 1996).

32.3.4 Rapid Prototyping and Iteration

The most direct way to learn about and respond to human needs and preferences is to observe them as they carry out activities related to the system or design concept under development; in other words, to use one or more of the empirical methods described in Sections 32.3.1 and 32.3.2. A key enabler of this approach is to build prototypes for testing, generally with the assumption that these prototypes will be built rapidly and in a way that enables iteration in response to feedback from users or other stakeholders.

As with other usability engineering methods, rapid prototyping can be conducted at differing levels of detail and veridicality with respect to the design vision. For instance, early in a project, one or more prototypes may be created using everyday office supplies such as cardboard, paper, and colored markers. This phase of usability engineering is often referred to as “paper prototyping.” Practitioners have documented a range of useful techniques for constructing these low-fidelity prototypes (Snyder, 2003; http://www.paperprototyping.com).

A low-fidelity prototype might be created to represent key display screens or dialog steps, and users can be recruited to respond and “act upon” the design concepts; the problems and successes they have would be noted and used to guide refinement of the underlying concepts and interaction techniques. In a focus group setting, the team might arrive with storyboards or multimedia animations of current design scenarios, presenting these to the group of users for reactions and suggestions. As the design concepts become more stable, the team’s software developers will begin to build software prototypes; this may be done in phases using different authoring tools with increasing fidelity, or in an evolutionary fashion where a single system is gradually expanded and refined until it is fully operational.

A number of usability engineering practitioners have debated and studied the impact of prototype fidelity in evaluating users’ performance and subjective reactions to a system under development (e.g., Nielsen, 1990; Sauer et al., 2008; Virzi et al., 1996). Many of these studies have shown that low-fidelity prototypes can evoke results from user tests very similar to those carried out with more veridical systems, particularly with respect to detecting and classifying the most severe problems that a team would almost certainly want to address (Sauer and Sonderegger, 2009). Interestingly, users seem to adjust their aesthetic evaluations to some extent when presented with “rough” prototypes, perhaps recognizing that these surrogates should not be judged according to the same criteria (Sauer and Sonderegger, 2009). Practitioners might be expected to prefer lower fidelity methods, when possible, simply because of the reduced cost of creating them and the simplicity of iteration (e.g., redrawing a screen). However, it is clear that when realistic and comprehensive measures of user experience and performance are required, a fully functioning prototype must be provided. This is unlikely to be available until relatively late in the design and development life cycle. Rudd et al. (1996) discuss the practical trade-offs between low versus high fidelity.

32.3.5 Cost-Justifying Usability in Real-World Settings

As was implied in the discussion of analytic methods and the trade-offs in low- versus high-fidelity prototyping, the costs and benefits of doing usability engineering is an important issue in the real-world settings of companies doing software development. At the most extreme, if the target user population will accept and adopt a piece of software that has horrible usability characteristics (i.e., because it fulfills
an essential function not available in any other way), why would a company spend time and money to analyze and support user interaction requirements? Even when it is clear that a product’s usability will influence people’s decision to buy and use it, how can a company justify an empirical study of its usability when the team’s usability inspection can yield a rich number of issues to consider and address? Why build a high-fidelity prototype at all?

Because of such concerns, researchers and practitioners over the years have attempted to weigh the costs and benefits of conducting usability evaluations. Jakob Nielsen pioneered this thread of work with his proposals for “discount usability engineering” and his development of the heuristic evaluation methods (Nielsen, 1994; Nielsen and Molich, 1990). He has provided a number of analyses that contrast different usability engineering methods, estimates for how many test users are needed to discover critical problems, the role of analyst expertise, and so on. Several detailed discussions can be found on the website he maintains as part of his usability consulting business (useit.com). For example, he summarizes the observed usability improvement (as indicated by task time and errors) across the iterative design of four different case studies, estimating that each new version led to an improvement of about 40% in usability (useit.com/iterative_design). However, Nielsen does not provide the corresponding costs of iterating the designs, nor does he consider the variability in improvement one would expect across different design projects and teams.

One of the first practitioners to describe the use of cost–benefit analysis for usability engineering methods was Claire-Marie Karat (1990). Such a calculation depends on first estimating the benefits, for example, the seconds saved by future users who carry out tasks with a piece of software that has been improved through user testing (an average savings of 4 s per e-commerce transaction for a task carried out one time per week by 100 K users can quickly add up to many dollars of saved customer time). These benefits are then compared to the costs of conducting the iterative design: the costs of one or more usability engineers to analyze tasks, develop test materials, recruit users and conduct test sessions, and so on; the cost of the test users’ time and associated testing equipment; the cost of responding to the user feedback, and so on. Of course, any such estimation process depends greatly on the assumptions made concerning the usability engineering methods to be used and their effectiveness in guiding redesign. A number of usability cost–benefit analyses developed by Karat and others can be found in the books edited by Bias and Mayhew (1994, 2005).

### 32.4 Current Challenges for Usability Engineering

Most of the methods for usability engineering emerged during the 1980s and 1990s, as computers became increasingly pervasive and important as personal productivity tools. During that era, the conventional view of a “user test” was a single user sitting at a single computer with single screen, working on a single-threaded task. Such a setting was quite amenable to engineering methods, because it could be structured, observed, and analyzed with considerable precision. However, the variety in interactive software and user interaction settings has increased enormously in the past two decades, creating tremendous challenges for usability engineers. We turn now to a discussion of trends in both technology and usage settings, along with corresponding implications for usability engineers working in these new arenas.

#### 32.4.1 New Technology Contexts for Usability

Users no longer work alone. Increasingly the presumption is that people are connected to others through their technology and that often their day-to-day activities depend on those connections. As computing has become more ubiquitous in our everyday lives, the devices and associated input and output channels that people use for technology interactions have expanded. Finally, user interaction dialogs have become more intelligent, largely as a result of expanding access to large datasets and a concomitant growth in computational power.
32.4.1.1 The Web and Network-Based Systems

The World Wide Web has become a pervasive element of many technology-mediated activities. For some tasks, the web provides critical support for communication with other people or institutions; for others, it is the broad access to information that is most important. Even software that is designed to operate without a network connection will often rely on the web for delivery of regular updates or other maintenance services. As a result, the networked information and services available through the web must be recognized and analyzed when usability engineers are studying whether and how users will interact with a given software system.

A particularly common way for application designers to leverage the web is in user help and information services (Lee and Lee, 2007). While it is possible to package a core set of information with the software itself, a much larger and more dynamic body of information can be provided through the web. Web-based reference information and other documentation can be maintained and expanded in a centralized and convenient fashion as the application is used and evolves, making the help available more extensive. An application’s users may also contribute content, perhaps by asking questions that become an index into the help provided, or by answering questions posed by other less-experienced users (Ackerman and Malone, 1990). More broadly, users regularly turn to the web at large for tips or guidance on how to solve a problem, even when the application offers an extensive help system of its own. Surveys of users’ preferences for learning about novel technologies often report strong ratings for either asking a colleague or finding an example solution (Rosson et al., 2005; Zang and Rosson, 2009).

One challenge faced by usability engineers is that the web and its services are extremely dynamic and may result in rather different experiences for different users; the dynamic nature of the web also gives designers significantly less control over the user experience. Although the designers of a web application may carefully architect its content, look, and feel, the boundary between the designed application and the rest of the web may still be quite blurred in users’ experiences. Considering our e-commerce example, it would be typical for the service to invite comments or reviews from customers, often with little oversight in the details of what and how comments are posted. It is also quite common for Internet applications to include links to other network content or applications, and it is not clear whether and how users integrate across applications when they follow such links.

32.4.1.2 Ubiquitous Computing

The mechanisms through which users interact with software have also expanded in the past two decades. While the most common context for use continues to be interaction with a personal computer using a mouse and keyboard, other options are increasingly common. Computing devices have decreased in size, and wireless connectivity to the Internet is available throughout the world through telephone or Wi-Fi networks.

As the variety in location and activity contexts of use have broadened, so have the options available for interacting with devices. Users who are moving around in the world may find it difficult to look at a small screen or to interact with mouse and keyboard. As a result, technologies that enable voice or touch input are becoming more common (Ishi, 2008; Vertanen and MacKay, 2010). Sensors built into the devices (e.g., a Global Positioning System for tracking location) can also provide useful information to an application that the individual no longer needs to input. However, adding an input channel that is beyond the user’s control can raise a new set of usability issues, for example, problems with maintaining user engagement and appropriate awareness of information collected over the external channel (Brown and Laurier, 2012; Leshed et al., 2008). Because ubiquitous use implies interactions with different devices or modalities depending on the context, system support for synchronizing personal data and application preferences across these different contexts has also become critical.

Ubiquitous computing has usability consequences that are similar to that of pervasive network access; that is, it becomes very difficult to predict exactly where, when, and how an application will be activated to carry out a task. Using a mobile application while driving has very different usability requirements.
and consequences than using the same functionality in one’s home or office (Williamson et al., 2011). Reading from a display that is outside in natural sunlight is a very different perceptual process than reading the same display under normal office lighting. These real-world, constantly changing usage contexts create considerable challenges for lab-based testing; so usability engineers must rely increasingly on fieldwork to evaluate the effectiveness of their designs.

### 32.4.1.3 Intelligent User Interfaces

A third general trend for interactive computing is the level of intelligence that software contains. With pervasive access to the Internet, an application can access and process a huge volume of data to “assist” a user with a task. This has become commonplace in e-commerce activities, where customer reviews and ratings are omnipresent, and where the bargains, prices, and even the presentation of products may be finely tuned to the person’s prior shopping history. Many online retailers provide social recommendations—guesses as to which items a user might find of interest based on the behavior of other users judged to be similar—to guide users’ shopping decisions. In this context, usability experts have worried about the types and amounts of information that is being tracked for individual users (Karat and Karat, 2010), even though most users count on these sorts of guidance.

Designers of learning support systems have long been interested in intelligent user interfaces. These include tutors that build and evolve a model of the learner’s current knowledge state so that the learning activities and feedback can be carefully matched to the person’s needs (London, 1992). But, like online recommendations, systems that adapt to a given user have become pervasive—even an application as commonplace as a web browser will now adapt to users’ location and language; personal mail clients “learn” about what is and is not junk mail; and cell phone battery apps adjust to the owner’s typical usage rhythms for energy-consuming features over a 24-hour period. Most of these intelligent adaptations are designed to improve usability and are usually easy to turn on and off. It is, however, still an open question whether an end user would be interested and willing to tune the adaptive model to make it even more effective (Kulesza et al., 2012).

Perhaps, the most salient example of intelligent user interfaces is the wide variety of robots that increasingly are being developed for special purpose activities such as cleaning (Marrone et al., 2002), information provision (Lee et al., 2010), or dangerous work environments like war zones or firefighting (Motard et al., 2010). Often, robots are designed as autonomous systems, but enough of them involve human interaction that an entirely new subarea within HCI has developed: human–robot interaction (HRI; see e.g., the Conference on Human–Robot Interaction series at http://humanrobotinteraction.org/category/conference/). Many of the usability issues in HRI are similar to those studied for years in HCI, for example, the impact and relative desirability of anthropomorphism, or the trade-offs in building affective features into a robot whose job is to answer questions. Thus, although robots may seem qualitatively different from an “interactive application,” in principle, HRI should follow the same usability life cycle used by other software projects—fieldwork that assesses how a robot might fit into or enhance current practices, prototyping and formative evaluation to envision and build the system, and summative evaluation to assess whether and how it meets stated requirements (Lee et al., 2009).

### 32.4.2 New Usage Contexts for Usability

In parallel with changing technologies for user interface development, the activity contexts in which computing takes place have changed significantly in the past two decades. Two aspects that are very relevant for usability engineering are the degree of collaboration and multitasking present in many computing activities.

#### 32.4.2.1 Collaborative Computing

Because users are often connected to a network while they engage in computing, there is a collaborative background to their work or play, even when the application does not explicitly support or encourage
interaction with other people—for instance, simply sending a document or image as an attachment through email represents a loosely coupled form of collaborative computing. When examining current practices as part of a design process, the consideration of an activity’s social or cultural context has become a prime concern along with the more traditional human factors stemming from users’ physical, perceptual, or cognitive capacities (Beyer and Holtzblatt, 1997; Rosson and Carroll, 2002).

When explicitly designing for collaborative use, usability engineers must address a broad range of factors that interact with the nature and types of interaction that will take place among collaborators. For example, an asynchronous collaborative activity that transpires over different locations and times has rather different needs than a synchronous interaction that connects remote collaborators in real time (Neale et al., 2004). The different options for computer-mediated communication bring different affordances for the style and quality of the exchange (Clark and Brennan, 1991), with text-based communication being adequate for simple information sharing but richer audiovisual channels important for collaborations that rely on evaluation and negotiation of options for decision-making (Monk, 2008). To further complicate the issue, users’ communication may change over time, as collaborators construct mental representations of one another that can guide and simplify their interactions (Convertino et al., 2011). Beyond the general concern for matching communication channels to collaborators’ needs is the more general requirement for helping users to stay aware of what collaborators have been doing, whether in real time or not (Carroll et al., 2006). For software intended to support global collaboration contexts, a special usability challenge is the conflicts that may arise when global teams use a mix of face-to-face (i.e., within a locale) and remote (i.e., across locales) interactions (Ocker et al., 2009).

### 32.4.2.2 Multitasking

Modern operating systems are designed to support multithreaded activities, allowing users to launch and interact with multiple applications at the same time. In a work setting, this might mean transitioning in and out of a mail client, a document editor, an analysis package, and so on. Sometimes, the decision to transition is initiated by the user, but at other times, a user is “called to attention” by a notification of an application’s status change. In general, the presence of multitasking does not raise new requirements for user interface design and usability engineering. Multitasking increases the importance of features that have long been in focus—for example, providing feedback during and after a task or subtask has been completed, and providing status information that conveys whether and how an interrupted task can be resumed (Norman, 1988). When designers support multitasking directly with complementary input and output channels, the situation becomes even more complex, because users must not only be able to track multiple goal–action sequences in parallel, but may also be required to orient to different configurations of displays or input devices for these different task contexts.

There are many usability engineering challenges that ensue from the complications of web-based platforms, pervasive connectivity and collaborative tasks, and the general trend toward greater and greater multitasking and heterogeneity in devices. To some extent, the challenges are simply those of greater complexity. For example, a meaningful reference task for evaluation may involve multiple parallel tasks, a group of users, or tasks that take place over extended periods of time and multiple locations (Neale et al., 2004). However, the new contexts also foreground new usability issues, for example, issues of privacy related to the persistent collection, analysis and repurposing of users’ personal data or online behavior, and issues of perceptual and motor “transfer” among rather different input and output devices used for the same tasks.

### 32.4.3 New Concepts and Methods for Usability

Early HCI focused on software systems to support work activity, typically the activity of single users in the workplace. Today, HCI addresses learning and education, leisure, and entertainment; it addresses people individually, in groups, and as mass society. HCI is now about coping with everyday life, citizenship and civic participation, and very much about satisfaction as a consumer. This diversification has evoked new problems in usability engineering, required new concepts and approaches, and has drawn
new people and skills into the field. The fundamental goals however remain the same: to design and refine software systems to meet human needs. This diversification is a natural consequence of growth and success, though it has created microcosms of specialized usability engineering work that are not always well coordinated with one another or with a broader conception. Nevertheless, these initiatives are driving usability engineering forward.

32.4.3.1 Universal Usability

Many of the most important innovations in display technologies or input devices have emerged from user interface design research directed at meeting the needs of people with disabilities. While the interest in accommodating people with special needs—whether with respect to visual, motor behavior, cognitive, or other capacities—has a long history, the resources and scientific interest aimed at these questions increased significantly in the 1990s, at least partly due to significant pieces of national legislation and policies (e.g., the Americans with Disabilities Act of 1990, the Telecommunications Act of 1996) that mandated accommodations in some work settings. The rapidly expanding population of elderly users in the United States has also helped to fuel research interest in these topics, with annual conferences such as ACM ASSETS and ACM Transactions on Accessible Computing created to enhance visibility of emerging findings and technology innovations.

A focus on universal usability—sometimes also called universal design—often produces findings or solutions that yield better designs for everyone. For example, the continued progress on voice commands helps not only the motor-impaired but also users who are in a hands-busy environment such as driving or working with mechanical equipment (Schütz et al., 2007). Alternate input devices such as joysticks or foot pedals are often integrated into custom HCI settings like games or training simulations (Zon and Roerdink, 2007). The many years of research on systems for text-to-speech translation not only support low-sight individuals (e.g., using screen readers like JAWS) but also is now omnipresent in voice-based telephone dialogs.

32.4.3.2 Interaction Design

As computer technologies have diversified, approaches to usability have broadened to consider the role and impacts of technology in consumer product design, for example, including furniture (Streitz et al., 1998), kitchen and other household tools (Chi et al., 2007), toys (Tomitsch et al., 2006), electronic books and games (Druin et al., 2009), and even clothing and accessories (Cho et al., 2009). Many such efforts fall under the umbrella of interaction design—often abbreviated as IxD—that differs from usability engineering in its pronounced emphasis on creative design thinking. That is, whereas in usability engineering the designers typically expect to begin with a detailed analysis of users’ current practices and associated requirements, IxD designers specialize in out-of-the-box imagination and exploration of how things might be without the constraints of the current world (Löwgren, 2008).

In the IxD vision, the aesthetic characteristics of a design concept are of equal importance to its technical or functional elements, and these characteristics are often tied to physical form of the product. For instance, Apple has developed a consistent and successful design aesthetic through its i-devices; people choose to purchase and use these products not just because they satisfy their task needs but because they like the way the objects look and feel. Because of the importance of the overall aesthetic, sketches and other tangible artifacts are significant means for communicating design ideas. Learning to create and elaborate these early design prototypes is an essential skill for IxD professionals (Buxton, 2007).

32.4.3.3 Experience Design

Similar to IxD, the emerging discipline of user experience design (UX) places great value on the subjective experience of people who will be interacting with technology. UX recognizes that this subjective experience is embedded in a complex network of related concerns, for example, the “family” of products associated with a particular offering, how a product’s users feel about the company that produces it, how often and by what means the product will be upgraded, its price relative to competitors, and so on (Diller et al., 2005).
The UX design process centers on the mechanisms for engagement. Design representations such as personas (Cooper, 1999; Grudin and Pruitt, 2002) are particularly popular within this design community, because a well-crafted persona offers a means of vividly evoking subjective experiences. Knowing that 32-year-old Jane Pelham loves to watch professional sports on TV, is an avid cook, and is carefully watching her weight may seem incidental to discussions about a new feature for an office product, but this relatively vivid depiction can help the product designers imagine how Jane would react to a design decision involving support for a touch-screen interface (e.g., how might she use it in the living room vs. in the kitchen). In this sense, UX can be seen as a more holistic version of the SBD process overviewed earlier—it relies centrally on representing and reasoning about individual user experiences rather than generic population statistics, but unlike SBD, the process does not entrain any systematic analysis and representation of design trade-offs and rationale.

### 32.5 Summary and Conclusions

To appropriate Robert Browning, usability engineering is an area of technical endeavor where our reach must necessarily exceed our grasp. New technology infrastructures, new applications, and new human aspirations drive usability engineering practices toward an expanding frontier of new concepts and techniques. Like many topics in computer science and engineering, we are not likely to ever close the book on usability engineering practices.

Why would we want to? Science and technology are most exciting at the frontier. The primary constant in usability engineering through four decades has been a developmental trajectory. Usability engineering has constantly become richer, more broadly applicable, more diverse conceptually and methodologically, and much more effective. The need for this trajectory to continue is unabated, and the prospects for further innovation and growth, driven by both external technological factors and human imagination, remain strong.

### References


Cooper, A. 1999. *The Inmates Are Running the Asylum*. Indianapolis, IN: SAMS.


