(Symantec Corporation), Brian Shackel (Loughborough University of Technology, U.K.), Ben Shneiderman (University of Maryland), Scott Stornetta (Bellcore), Kurt Sussman (Symantec Corporation), Desirée Sy (Information Design Solutions), Michael Tauber (University of Paderborn, Germany), Bruce Tognazzini (SunSoft), Hirotada Ueda (Hitachi Central Research Laboratory and FRIEND21 Research Center, Japan), Gerrit van der Veer (Free University of Amsterdam, The Netherlands), Floris L. van Nes (Institute for Perception Research/Philips Research Laboratories, The Netherlands), Robert Virzi (GTE Laboratories), Christopher A. White (GTech Corporation), Richard Wolf (Lotus Development Corporation), and Peter Wright (University of York, U.K.).

The resulting book is solely the responsibility of the author, and the people mentioned above should not be held responsible for the way I have interpreted their comments and advice. Most of this book was written while I was on the applied research staff of Bellcore but it should not be taken as necessarily representing any official views or policies of Bellcore.

This printing of the book has been updated with a number of literature references and comments on developments since the book was first published. Among other things, I added the new synergy review method, which I invented just a few weeks after sending in the final copy for the first printing. Mostly, the book is unchanged, though, since the basics of usability engineering remain fairly constant and do not vary from year to year.

Jakob Nielsen
Mountain View, California
April 1994

Chapter 1 *Executive Summary*

Have you ever seen one of the people who will be users of your current project?[1] Have you talked to such a user? Have you visited the users' work environment and observed what their tasks are, how they approach these tasks, and what pragmatic circumstances they have to cope with? Such simple user-centered activities form the basis of usability engineering. More advanced methods exist and are covered later in this book, but just a simple field trip to observe users in their own environment working on real-world tasks can often provide a wealth of usability insights.

In one example, three one-day visits to branch offices of a medium-sized insurance company produced a list of 130 usability problems [Nielsen 1990b]. The system design was sound, and most of the problems were simple enough to fix once they were known (but, of course, they would not have been known if it had not been for the field study). Many of the 130 items were serious problems only for novice users. However, even very experienced users were estimated to waste at least 10 minutes every day because of usability

1. Note that you have to talk to the individuals who will be using the system. Talking to the users' manager or vice president for data processing does not count since these people are likely to have a completely different understanding of the job than the actual users.

problems, costing the company large amounts of money in both labor costs and lost sales opportunities.

The staff was often interrupted by telephone calls or walk-in clients. Unfortunately, several subsystems were not designed for interruptions—users lost all of their work if a transaction was not carried to completion. At one small branch, an agent stated that she never used the damage-claims subsystem during periods where she was the only person in the office and had to answer all calls. In some cases, agents were observed using other agents' terminals (and "borrowing" their passwords) to deal with interruptions rather than quit one of the unforgiving subsystems in the middle of a transaction.

In another case, the system allowed only one line for error messages, so it had to give an obscure, truncated version of a long message. The full message was available by pressing the help key, PF1, an action the developers in the central data-processing office felt was very natural. But users in the branch office had not made the conceptual leap that told them the help key was doubling as an extended-error-message key. Instead, they wasted a lot of time trying to understand the truncated message. A better design would have used the one line on the screen for a brief indication of the error, followed by "`PF1 for more information`" or a similar instruction.

1.1 Cost Savings

There are several well-documented examples of cost savings from the use of usability engineering methods.[2] For example:

- When a certain rotary dial telephone was first tested, users were found to dial fairly slowly. A human factors expert spent one hour to come up with a simple graphical interface element which speeded up users' dialing behavior by about 0.15 seconds per digit, for a total annual saving of about $1,000,000 in reduced demands on the central switches [Karlin and Klemmer 1989].
- An Australian insurance company had annual savings of A$536,023 from redesigning its application forms to make customer errors less likely [Fisher and Sless 1990]. The cost of the usability project was less than A$100,000. The old forms were so difficult to fill in that they contained an average of 7.8 errors per form, making it necessary for company staff to spend more than one hour per form repairing the errors.
- A major computer company saved $41,700 the first day the system was in use by making sign-on attempts faster for a security application. This increased usability was achieved through iterative design at a cost of only $20,700 [Karat 1990].
- The 25 "human factors success stories" discussed by Harris [1984] include the improvement of the Boeing 757 flight deck interface to allow operation by two instead of three pilots, the 35% increase in alignment speed in a production line for integrated circuits, the reduction from 3,000 words to 150 words of instructions needed to operate a paging device, and even an improvement in a drunk-driver detection system that increased the arrest rate per police officer patrol-hour by 12%.

Unfortunately, the cost savings from increased usability are not always directly visible to the development organization since they may not show up until after the release of the product. As an extreme example, Fisher and Sless [1990] report that the Australian government can process a tax return for A$2.25 on the average. At the same time, the average Australian resident spends 11 hours filling in the form, and 62% of Australians have to use agents to help do the job. If the complexity of the tax forms were reduced, these "customers" might therefore realize huge savings in time and advisor fees, but the government might only save a few cents in processing costs. In the same way, making a spreadsheet easier to learn might only save the vendor a small amount in reduced hotline staffing levels, even though each customer might save several hours of unnecessary work.

2. There are more examples of cost savings that are less well documented. As noted by Chapanis [1991], most case studies fail to meet the rigorous methodological requirements that are necessary to be absolutely sure what cost savings can be attributed to user interface improvements since there are often several other changes made simultaneously (e.g., [Thompson *et al.* 1986]).

Distributed benefits of a few hours per user are hard to measure and do not immediately add up to hard cash [Sassone 1987]. For example, redesigning the interface to an oscilloscope increased user productivity by 77% during the time they were using the scope [Bailey *et al.* 1988], but the productivity impact on the total workday of an engineer was much less dramatic and therefore had less impact. The customers *do* save with better interfaces, though, and these savings presumably translate into a better reputation for the product and therefore eventually increase sales. Unfortunately, the effect of having increased usability lead to increased sales has mostly been documented only anecdotally.[3] In several cases, the relative usability of competing products is well known in the industry, and computer salespersons often recommend certain software packages on the basis of their usability.

Because much of the financial payoff from usability methods shows up after the release of the product, some usability specialists [Grudin *et al.* 1987] have advocated shifting parts of the responsibility for usability engineering toward middle and upper management levels instead of the development managers. Even the development manager may see some immediate benefits from usability engineering, however, in the frequent case when early usability studies reveal that there is no need for certain contemplated features. If users' needs are not known, considerable development efforts may be wasted on such features in the mistaken belief that some users may want them. Users rarely complain that a system can do too *much* (they just don't use the superfluous features), so such over-design normally does not become sufficiently visible to make the potential development savings explicitly known. They are there nevertheless.

3. In one of the few documented cases, a usability study of the first version of a fourth-generation database system revealed 75 usability problems. Twenty of the most serious problems were fixed in the second release, which generated 80% higher product revenues than the first release [Wixon and Jones 1994]. This revenue increase was 66% higher than sales projections and so is probably due to the improvements in usability since field test customers were reported to point to the user interface as the most significant improvement in the product.

Usability studies can often be conducted very quickly and with small budgets. For example, Bailey [1991] gives an example of a study that required no more than five and a half hours to find out that the addition of color did not help users in a certain menu selection task. A development group could easily have spent many more staff hours arguing over this design issue in meetings than it took to resolve it by testing.

A study of software engineering cost estimates showed that 63% of large software projects significantly overran their estimates [Lederer and Prasad 1992]. When asked to explain their inaccurate cost estimates, software managers cited 24 different reasons and, interestingly, the four reasons that were rated as having the highest responsibility were all related to usability engineering: frequent requests for changes by users, overlooked tasks, users' lack of understanding of their own requirements, and insufficient user–analyst communication and understanding. Proper usability engineering methodology will prevent most such problems and thus substantially reduce cost overruns in software projects.

Even though the use of usability methods definitely involves some benefits, some managers might hesitate to use them because of their perceived high cost and complexity. For example, a paper in the widely read and respected journal *Communications of the ACM* estimated that the "costs required to add human factors elements to the development of software" was $128,330 [Mantei and Teorey 1988]. This sum is several times the total budget for usability in most smaller companies, and one interface evangelist has actually found it necessary to warn such small companies against believing this estimate [Tognazzini 1990]. Otherwise, the result could easily be that a project manager would discard any attempt at usability engineering in the belief that the project's budget could not bear the cost. Luckily, usability projects can easily be completed at substantially lower budgets as discussed in the section on Discount Usability Engineering (page 16). This entire book is aimed at presenting usability methods that can be used no matter what budget is available.

	Bottom Quartile (Q_1)	Median (Q_2)	Top Quartile (Q_3)
Project size in person-year	11	23	58
Actual usability budget relative to total	4%	6%	15%
Ideal usability budget relative to total	6%	10%	21%
Actual usability effort in person-years	1.0	1.5	2.0
Ideal usability effort in person-years	1.7	2.3	3.8

Table 1 *Results from a survey of the usability budgets of 31 development projects that had usability engineering activities.*

In January 1993, I surveyed 31 development projects that had usability engineering activities to find how much of their budget was devoted to usability. Respondents were also asked to estimate how large the usability budget ideally should be for their project. The results are shown in Table 1 and indicate that usability accounted for about 6% of the budgets for these projects and that the respondents felt that the ideal usability budget would have been 10%. The ideal desired usability effort in person-years is essentially independent of project size ($r = .12$) when three very large outlier projects of 250–350 person-years are excluded from the analysis. This result makes some sense in that many usability activities take about the same time to perform, no matter how difficult the program is to implement. Very large systems may have more elements in their user interfaces (screens, dialogue boxes, menus, etc.), meaning that they may need somewhat more time for usability activities, though definitely not proportionally more. The main conclusion to be derived from Table 1 is thus that two person-years of usability engineering is the median effort to aim for in a project and that four person-years would be sufficient for most projects. Of course, the actual amount of usability work needed in a project will depend on the characteristics of the project.

In a study of several corporations, Wasserman [1989] found that many leading companies allocated about 4–6% of their research

and development staff to interface design and usability work. He believes that 2% is the critical lower limit for designing competitively effective products but acknowledges that many companies are significantly below that level. There is not necessarily a conflict between Wasserman's findings and Table 1. First, the survey in Table 1 is more recent, and usability has increased in importance in recent years.[4] Second, my survey only involved those projects in the various companies that have active usability engineering efforts, and many companies still have some projects without any usability engineering activities, which would contribute to making their overall usability budgets smaller than the usability budgets for those projects that had usability activities.

The final budget recommendation for any given product or company would of course depend on the specific nature of the projects. If a product is aimed at the population at large, then a substantial usability effort is probably necessary to ensure broad acceptance of the product. Similarly, a product that is going to see substantial daily use in a business can also cost-justify a large usability investment from the expected savings. And finally, one would normally recommend a rather limited[5] usability effort for systems that are only going to be used by a small number of highly skilled and trained users.[6]

When considering usability budgets, remember that your system *will* be tested for usability even if you don't do so yourself. Your customers will do it for you, as they struggle to use the system. Any usability problems found by users in the field will undermine

4. A 1971 paper estimated that a reasonable level for usability budgets for non-military systems was about 3% [Shackel 1971], providing some additional evidence that usability has increased in budget share over the years.

5. "Limited" does not mean "nonexistent," however. There is *always* a payoff to be gained from applying a few, cheap usability methods.

6. For certain applications such as a tactical support system for a fighter pilot or the control room for an expensive or dangerous plant, small differences in user performance can be a matter of life or death, and major usability efforts may be called for even though the users are few and highly trained. Two thirds of aircraft accidents can be attributed to the cockpit crew rather than to equipment malfunction and thus could potentially have been avoided with better human factors [Nagel 1988].

your reputation for quality products and the resulting change requests will be about 100 times more expensive to implement than changes discovered by yourself in the early phases of the project.

1.2 *Usability Now!*[7]

User interfaces are now a much more important part of computers than they used to be. The revolution in personal computers and falling hardware prices are making computers available to ever broader groups of users, and these users are using computers for a larger variety of tasks. When computers were only used by a small number of people who mostly performed very specialized tasks, it made sense to require a high degree of learning and expertise of the users. Also, computers were once so expensive that it was reasonable to let users suffer a little if the computer could be utilized more efficiently. Now it pays to dedicate a large proportion of the computational resources (CPU cycles, memory use, communication bandwidth, screen space, development effort) to "nothing else" than making life easier for the user.

Furthermore, video games and some of the better personal computer software have shown users that it is possible to produce pleasant and approachable interfaces, so they are becoming much less willing to suffer from low usability. *Business Week* had a cover story in 1991 entitled *"I Can't Work This ?#!!@ Thing!"* [Nussbaum and Neff 1991], reporting on consumer dissatisfaction with overcomplex interfaces on video recorders and other gadgets and that several major companies were redesigning their products to make them easier to use.

Time itself is on the side of increasing the perceived need for usability since the software market seems to be shifting away from the "features war" of earlier years [Telles 1990]. User interface design and customer service will probably generate more added

7. After writing my first draft of this section, I learned that the U.K. Department of Trade and Industry (DTI) has a research and development program that also is called *Usability Now!* [Wiggins 1991]. So much the better.

value for computer companies than hardware manufacturing [Rappaport and Halevi 1991], and user interfaces are a major way to differentiate products in a market dominated by an otherwise homogenizing trend toward open systems. Now most software products have more features than users will ever need or learn, and Telles [1990] states that the "interface has become an important element in garnering good reviews" of software in the trade press.[8] In an unpublished study from 1990, Tim Frank Andersen from the Technical University of Denmark read 70 reviews of software products in various personal computer magazines and counted 784 comments on the usability of the reviewed software. This is an average of 11.2 usability comments per software review. Now, many of these comments were fairly superficial, but their sheer number indicates the importance of usability in today's marketplace. If anything, usability has increased in importance in software reviews since the 1990 study to the extent that some personal computer magazines now have usability laboratories for use in comparative testing of software products and include usability statistics like average task times in their reviews [Reed 1992].

Recently, we have even seen political demands for regulations with respect to usability. Currently, most such political initiatives seem to be directed toward hardware ergonomics, but some also include software usability. As further discussed in Chapter 8, several international user interface standards are currently being developed. These standards may well gain the force of law in certain countries and will certainly have great impact in many other countries [Stewart 1990]. The European Union has passed a directive[9] on work with display screens stating that since December 31, 1992,

8. The trade magazine *InfoWorld* assigns explicit weights in its software reviews: Ease of learning is weighed at 4–10%, ease of use at 8–13%, and quality of documentation at 5–8%, with the exact weights determined by the type of application being reviewed. These three review criteria account for between 18% (spreadsheets) and 30% (word processors) of the final review scores. Error handling is assigned a further 5–8% of the review weight as a combined category including both user errors (a usability issue), software bugs, and recovery from hardware crashes. Notice, by the way, how favorably these percentages compare with the allocation of 6–10% of a development project's budget to usability mentioned on page 6; usability is a comparatively cheap way to improve product quality.

- Software must be suitable for the task
- Software must be easy to use
- The principles of software ergonomics must be applied

for all display screen workstations (computers and such) put into service in the E.U. Even though the requirements are very general in nature, they still indicate the direction of the political pressure for increased usability.

1.3 *Usability Slogans*

Major parts of the usability approach in this book can be summarized in the short slogans given here. You will find that some of the slogans contradict each other. Unfortunately, usability is filled with apparent contradictions that are only resolved after more detailed analysis. Some contradictions and trade-offs will always remain, and it is the job of the usability engineer to arrive at the best solution for the individual project's needs. There are very few hard and firm rules in usability that do not have some exceptions. For the full story, read on.

Your Best Guess Is Not Good Enough

A basic reason for the existence of usability engineering is that it is impossible to design an optimal user interface just by giving it your best try. Users have infinite potential for making unexpected misinterpretations of interface elements and for performing their job in a different way than you imagine.

Your design will be much better if you work on the basis of an understanding of the users and their tasks. Then, by all means design the best interface you can, but make sure to validate it with user tests and the other methods recommended in this book. It is no shame to have to revise a user interface design as a result of user

9. Council Directive of May 29, 1990, on the minimum safety and health requirements for work with display screen equipment (90/270/EEC), *Official Journal of the European Communities* No. **L 156**, 21.6.1990, 14–18.

testing. This happens to the best of usability experts, and it might indeed be a true measure of usability maturity that one is willing to acknowledge the need to modify initial design choices to accommodate the users. Julius Caesar is widely acknowledged as one of the greatest generals of antiquity. Even so, his true talent was not perfect campaign planning but his ability to adjust to the situation as it evolved. He often placed his legions in highly problematic situations which they only survived because he changed his plans to accommodate the facts. If Caesar could conquer France by admitting his mistakes [Caesar 51 B.C.], then maybe you can win some market share by admitting yours.

The User Is Always Right

As mentioned, all experience shows that any initial attempt at a user interface design will include some usability problems. Therefore, the user interface developer needs to acquire a certain design humility and acknowledge the need to modify the original design to accommodate the user's problems. The designer's attitude should be that if users have problems with an aspect of the interface, then this is not because they are stupid or just should have tried a little harder. Somebody once tested the usability of a user manual and found that users almost always made a mistake in a certain step of a particular procedure. Their solution was to frame the difficult step in a box and add a note saying *"Read these instructions carefully!"* Of course, the correct conclusion would have been that the description was too difficult and should be rewritten.

The User Is Not Always Right

Unfortunately, it does not follow that user interface designs can be derived just by asking users what they would like. Users often do not know what is good for them. One example is a study of the weight of telephone handsets conducted in the 1950s when people were used to fairly heavy handsets. The result of asking users whether they would like lighter handsets was no, they were happy with the handsets they had [Karlin and Klemmer 1989]. Even so, a test of handsets that looked identical but had different weights showed that people preferred handsets with about half the then-normal weight.

Users have a very hard time predicting how they will interact with potential future systems with which they have no experience. As another example, 73% of the respondents in a survey of 9,652 commuters said that they would not use a proposed information service with continual up-to-the-minute traffic information. But after they were shown sample screens from a prototype of the service, 84% of the respondents said that they would in fact use it [Gray *et al.* 1990].

Furthermore, users will often have divergent opinions when asked about details of user interface design. For example, studies of how people name things [Furnas *et al.* 1987] have shown that the probability of having two people apply the same name to an object is between 7% and 18%, depending on the object, clearly making it infeasible to design command names just by asking some user.

Users Are Not Designers

The ideal solution to the usability question might be to leave the design of the interface up to the individual users. Just provide sufficient customization flexibility, and all users can have exactly the interface they like. Studies have shown, however, that novice users do not customize their interfaces even when such facilities are available [Jørgensen and Sauer 1990]. One novice user exclaimed, "I didn't dare touch them [the customization features] in case something went wrong." Therefore, a good initial interface is needed to support novice users. Expert users (especially programmers) do use customization features, but there are still compelling reasons not to rely on user customization as the main element of user interface design.

First, customization is easy only if it builds on a coherent design with good previously designed options from which to choose. Second, the customization feature itself will need a user interface and will thus add to the complexity of the system and to the users' learning load. Third, too much customization leads each user to have a wildly different interface from the interfaces used by other users. Such interface variety makes it difficult to get help from colleagues, even though that is the help method rated highest by

both novice and expert users [Mack and Nielsen 1987]. And fourth, users may not always make the most appropriate design decisions.

For example, Grudin and Barnard [1985] compared command abbreviations they defined with abbreviations defined by individual users, and found that users made about twice as many errors when using their own abbreviations. Even when given the chance to redefine their abbreviations after the experiment, six of seven test users kept their poor abbreviation sets virtually intact, typically explaining that while yes, they had some problems with it, it seemed as good as any other set they could think of. Of course, users have other jobs and do not work as user interface professionals.

Designers Are Not Users

System designers are human and they certainly use computers: both characteristics of users. Therefore, it can be tempting for designers to trust their own intuition about user interface issues, since they do share these two important characteristics of the real users. Unfortunately, system designers are different from users in several respects, including their general computer experience (and enthusiasm) and their knowledge of the conceptual foundation of the design of the system. When you have a deep understanding of the structure of a system, it is normally easy to fit a small extra piece of information into the picture and interpret it correctly. Consequently, a system designer may look at any given screen design or error message and believe that it makes perfect sense, even though the same screen or message would be completely incomprehensible to a user who did not have the same understanding of the system.

Knowing about a system is a one-way street. One cannot go back to knowing nothing. It is almost impossible to disregard the information one already knows when trying to assess whether another piece of information would be easy to understand for a novice user. Landauer [1988b] uses a hidden animal picture as an analogy for developers' understanding of their own system. Hidden animal games, as well as popular children's books of the type, "*Where is so-and-so?*" show images with various levels of details, among which

is the animal or character one is supposed to find. Initially, it is very difficult to pick the animal out of the background, but once you have seen where the animal is, it is very easy to see it again. In fact, it is impossible to ignore one's knowledge of where the animal is and regain a perspective on the picture where one would have to search to find the animal.

A survey of 2,000 adults in Oregon showed that only 18% could use a bus schedule to find the time of departure [Egan 1991]. This finding does not indicate that the remaining 82% of Oregonians are less intelligent and should never be allowed on a bus.[10] Instead, the likely explanation is that the bus schedule was designed by people with extensive knowledge of buses and local transportation who just *knew* the meaning of every element on the schedule, and therefore never considered that parts of it might be difficult to understand for people who rarely take a bus.

Vice Presidents Are Not Users

Many CEOs and other top corporate executives have started to realize that usability is becoming one of their main competitive parameters, as user interfaces account for a steadily higher proportion of the value added in their products and services [Sculley 1992]. The downside of this higher visibility for user interfaces is that these executives may start meddling in user interface design.

Vice presidents and other corporate executives should realize that they are no more representative of the end users than the developers are. With the possible exception of management information systems and other software *intended* for vice presidents, corporate executives in a high-tech company are very different from the average user, and their intuitions about what would make a great design may not be accurate.

Boies *et al.* [1985] report that they sometimes had "a powerful person" in their company propose changes to their interface. They avoided making these changes by pointing out that this person

10. The same test showed that 97% of Oregonians could read a newspaper article and that 96% could tabulate two entries on a bank deposit slip.

probably had very different characteristics than the intended users and that their design had been tested on such real users. Of course, all design suggestions should be welcomed in order to serve as inspiration, but one should never be unduly swayed by a comment from a single person. People get promoted to vice president because of their managerial and decision-making skills, not because of their design skills.

Less Is More

One tempting solution to the user interface design problem might be to throw in any imaginable option or feature. If everything is there, then everybody should be satisfied, right? Wrong. Every single element in a user interface places some additional burden on the user in terms of having to consider whether to use that element. Having fewer options will often mean better usability because the users can then concentrate on understanding those fewer options [Brooks 1975]. Software reviewers are becoming aware that more features are not always better, and a major popular computer magazine ran a cover story lamenting the tendency of some programs to double in size every two years, coining the term "fatware" to describe bloated software [Perratore et al. 1993]. See also the discussion of the "less-is-more" principle on page 120.

Details Matter

Unfortunately, usability often depends on minor interface details, which is why systematic usability engineering work is necessary to ferret out those details. For example, Simonelli [1989] reports on the development of instructions for a frozen-dinner microwave indicator that would gradually change from being white to being blue. User testing showed that the phrase "turns blue" was much poorer than "white disappears" for describing this change, even though the two phrases are logically equivalent relative to this process. The blue color was not uniform—it was dark blue in some places and light blue in others— so users were uncertain "how blue is blue?" when the first wording was used.

Help Doesn't

Sometimes, online help and documentation doesn't really help the users [Mack *et al.* 1983]. That is to say, users often do not find the information they want in the mass of possible help and documentation and, even if they do find it, they may misinterpret the help. Also, help adds an extra set of features to a system, thus complicating the interface just by virtue of existing. In any case, the possibility for providing help should not been seen as an excuse to design a needlessly complex interface. It is always better if users can operate the system without having to refer to a help system. Usability is not a quality that can be spread out to cover a poor design like a thick layer of peanut butter,[11] so a user-hostile interface does not get user-friendly even by the addition of a brilliant help system. See also the section on Help and Documentation (page 148).

Usability Engineering Is Process

Most of this book consists of advice for activities to perform as part of the system development process. Readers may sometimes lose patience and wish that I had just told them about the *result* rather than the process: What makes an interface good? Unfortunately, so many things sometimes make an interface good and sometimes make it bad that any detailed advice regarding the end product has to be embellished with caveats, to an extent that makes it close to useless, not least because there will often be several conflicting guidelines. In contrast, the usability engineering *process* is well established and applies equally to all user interface designs. Each project is different, and each final user interface will look different, but the activities needed to arrive at a good result are fairly constant.

11. The peanut butter metaphor for misapplied usability engineering has been attributed to Clayton Lewis.

1.4 *Discount Usability Engineering*

Usability specialists will often propose using the best possible methodology. Indeed, this is what they have been trained to do in most universities. Unfortunately, it seems that *"Le mieux est l'ennemi du bien"* (the best is the enemy of the good) [Voltaire 1764] to the extent that insisting on using only the best methods may result in using no methods at all. Developers and software managers are sometimes intimidated by the strange terminology and elaborate laboratory setups employed by some usability specialists and may choose to abandon usability altogether in the mistaken belief that impenetrable theory is a necessary requirement for usability engineering [Bellotti 1988]. Therefore, I focus on achieving "the good" with respect to having *some* usability engineering work performed, even though the methods needed to achieve this result may not always be the absolutely "best" method and will not necessarily give perfect results.

It will be easy for the knowledgeable reader to dismiss the methods proposed here with various well-known counter-examples showing important usability aspects that will be missed under certain circumstances. Some counter-examples are no doubt true and I do agree that better results can be achieved by applying more careful methodologies. But remember that such more careful methods are also more expensive—often in terms of money and always in terms of required expertise (leading to the intimidation factor discussed above). Therefore, the simpler methods stand a much better chance of actually being used in practical design situations, and they should thus be viewed as a way of serving the user community.

The "discount usability engineering" [Nielsen 1989b, 1990b, 1994c] method is based on the use of the following four techniques:

- User and task observation
- Scenarios
- Simplified thinking aloud
- Heuristic evaluation

First, the basic principle of early focus on users should of course be followed. It can be achieved in various ways, including simple visits to customer locations. The main rules for "discount task analysis" are simply to observe users, keep quiet, and let the users work as they normally would without interference.

Scenarios

Scenarios are an especially cheap kind of prototype. The entire idea behind prototyping is to cut down on the complexity of implementation by eliminating parts of the full system. Horizontal prototypes reduce the level of functionality and result in a user interface surface layer, while vertical prototypes reduce the number of features and implement the full functionality of those chosen (i.e., we get a part of the system to play with).

Scenarios are the ultimate reduction of both the level of functionality and of the number of features: They can only simulate the user interface as long as a test user follows a previously planned path. See Figure 9 (page 94).

Since the scenario is small, we can afford to change it frequently, and if we use cheap, small thinking-aloud studies, we can also afford to test each of the versions. Therefore, scenarios are a way of getting quick and frequent feedback from users.

Scenarios can be implemented as paper mock-ups [Nielsen 1990d] or in simple prototyping environments [Nielsen 1989a], which may be easier to learn than more advanced programming environments [Nielsen *et al.* 1991]. This is an additional savings compared to more complex prototypes requiring the use of advanced software tools.

Simplified Thinking Aloud

The thinking-aloud method is discussed further in Section 6.8. Basically, it involves having one test user at a time use the system for a given set of tasks while being asked to "think out loud." By verbalizing their thoughts, users allow an observer to determine not just *what* they are doing with the interface, but also *why* they

are doing it. This additional insight into a user's thought process can help pinpoint concrete interface elements that cause misunderstandings, so that they can be redesigned.

Traditionally, thinking-aloud studies are conducted with psychologists or user interface experts as experimenters who videotape the subjects and perform detailed protocol analysis. This kind of method is certainly intimidating for ordinary developers. Those developers who have used the thinking-aloud method seem [Jørgensen 1989] to be happy with it, however. My studies [Nielsen 1992a] show that computer scientists are indeed able to apply the thinking-aloud method effectively to evaluate user interfaces with a minimum of training and that even methodologically primitive experiments will succeed in finding many usability problems.

Another major difference between simplified and traditional thinking aloud is that data analysis can be done on the basis of the notes taken by the experimenter instead of by videotapes. Recording, watching, and analyzing the videotapes is expensive and takes a lot of time that is better spent on running more subjects and on testing more iterations of redesigned user interfaces. Videotaping should only be done in those cases (such as research studies) where absolute certainty is needed. In discount usability engineering we don't aim at perfection; we just want to find most of the usability problems. A survey of 11 software engineers [Perlman 1988] found that they rated simple tests of prototypes as almost twice as useful as video protocols.

Heuristic Evaluation

Current collections of usability guidelines typically have on the order of a thousand rules to follow and are therefore seen as intimidating by developers. For the discount method I advocate cutting the complexity by two orders of magnitude, to just 10 rules, relying on a small set of broader heuristics such as the basic usability principles listed in Table 2 and discussed in Chapter 5, Usability Heuristics.

These principles can be used to explain a very large proportion of the problems one observes in user interface designs. Unfortunately,

- *Simple and natural dialogue:* Dialogues should not contain information that is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility. All information should appear in a natural and logical order.
- *Speak the users' language:* The dialogue should be expressed clearly in words, phrases, and concepts familiar to the user, rather than in system-oriented terms.
- *Minimize the users' memory load:* The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.
- *Consistency:* Users should not have to wonder whether different words, situations, or actions mean the same thing.
- *Feedback:* The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.
- *Clearly marked exits:* Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue.
- *Shortcuts:* Accelerators—unseen by the novice user—may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users.
- *Good error messages:* They should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.
- *Prevent errors:* Even better than good error messages is a careful design that prevents a problem from occurring in the first place.
- *Help and documentation:* Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, be focused on the user's task, list concrete steps to be carried out, and not be too large.

Table 2 *These usability principles should be followed by all user interface designers. This specific list was developed by the author and Rolf Molich [Molich and Nielsen 1990], but it is similar to other usability guidelines. See [Nielsen 1994d] for several lists of similar heuristics.*

it does require some experience with the principles to apply them correctly in all cases. On the other hand, even nonexperts can find many usability problems by heuristic evaluation, and many of the remaining problems would be revealed by the simplified thinking-aloud test. It can also be recommended to let several different people perform a heuristic evaluation as different people locate different usability problems.

1.5 Recipe For Action

As discussed on page 8, the demand for usability is growing rapidly these years. This book presents many steps that can be taken to increase usability. The most important advice to remember is that usability does not appear just because you wish for it. Get started on a systematic approach to usability—the sooner, the better. From a management perspective, the action items are

1. Recognize the need for usability in your organization.
2. Make it clear that usability has management support (this includes promoting a culture where it is seen as positive for developers to change their initial design ideas to accommodate demonstrated user needs).
3. Devote specific resources to usability engineering (you can start out small, but you need a minimal amount of *dedicated* resources for usability to make sure that it does not fall victim to deadline pressures).
4. Integrate systematic usability engineering activities into the various stages of your development lifecycle (see Chapter 4), including the early ones.
5. Make sure that all user interfaces are subjected to user testing.

If you think this 5-step plan is too much, then try this 1-step plan for a start:

1. Pick one of your existing user interfaces. Subject it to a simple user test by defining some typical test tasks, getting hold of a few potential customers who have not used the system before, and observing them as they try performing the tasks with the system (without *any* help or interference from you). If no usability problems are found, then be happy that you have been lucky. In the more likely case that problems are found, you already have your first usability project defined: get rid of them in the next release by using iterative design.