Chapter 6 *Usability Testing*

User testing with real users is the most fundamental usability method and is in some sense irreplaceable, since it provides direct information about how people use computers and what their exact problems are with the concrete interface being tested. Even so, the methods discussed in other chapters of this book can serve as good supplements to gather additional information or to gain usability insights at a lower cost.[1]

There are several methodological pitfalls in usability testing [Holleran 1991; Landauer 1988b], and as in all kinds of testing one needs to pay attention to the issues of reliability and validity. Reliability is the question of whether one would get the same result if the test were to be repeated, and validity is the question of whether the result actually reflects the usability issues one wants to test.

1. There is an association of usability professionals that publishes a newsletter on practical issues related to usability testing and meets at regular intervals to discuss issues related to usability. For further information about the Usability Professionals' Association, contact its office: Usability Professionals' Association, 10875 Plano Road, Suite 115, Dallas, TX 75238, USA. Tel. +1-214-349-8841, fax +1-214-349-7946.

Reliability

Reliability of usability tests is a problem because of the huge individual differences between test users. It is not uncommon to find that the best user is 10 times as fast as the slowest user, and the best 25% of the users are normally about twice as fast as the slowest 25% of the users [Egan 1988]. Because of this well-established phenomenon, one cannot conclude much from, say, observing that User A using Interface X could perform a certain task 40% faster than User B using Interface Y; it could very well be the case that User B just happened to be slower in general than User A. If the test was repeated with Users C and D, the result could easily be the opposite. For usability engineering purposes, one often needs to make decisions on the basis of fairly unreliable data, and one should certainly do so since *some* data is better than *no* data. For example, if a company had to choose between Interfaces X and Y as just discussed, it should obviously choose Interface X since it has at least a little bit of evidence in its favor. If several users have been tested, one could use standard statistical tests[2] to estimate the significance of the difference between the systems [Brigham 1989]. Assume, for example, that the statistics package states that the difference between the systems is significant at the level $p = .20$. This means that there is a 20% chance that Y was actually the best interface, but again one should obviously choose X since the odds are 4 to 1 that it is best.

Standard statistical tests can also be used to estimate the confidence intervals of test results and thus indicate the reliability of the size of the effects. For example, a statistical claim that the 95% confidence interval for the time needed to perform a certain test task is 4.5 ± 0.2 minutes means that there is a 95% probability that the true value is between 4.3 and 4.7 (and thus a 5% probability that it is really smaller than 4.3 or larger than 4.7). Such confidence intervals are important if the choice between two options is dependent not

2. Statistics is of course a topic worthy of several full books in its own right. See, for example, [Pedhazur and Schmelkin 1991] for basic methods, and [Lehmann and D'Abrera 1975] for more specialized statistical tests. An introduction to the use of statistics in the user interface field is given in Section 42.5 of [Landauer 1988b].

just on which one is best but also on how *much* better it is [Landauer 1988a, 1988b]. For example, a usability problem that is very expensive to fix should only be fixed if one has a reasonably tight confidence interval showing that the problem is indeed sufficiently bothersome to the users to justify the cost.

In a survey of 36 published usability studies, I found that the mean standard deviation was 33% for measures of expert-user performance (measured in 17 studies), 46% for measures of novice-user learning (measured in 12 studies), and 59% for error rates (measured in 13 studies). In all cases, the standard deviations are expressed as percent of the measured mean value of the usability attribute in question. These numbers can be used to derive early approximations of the number of test users needed to achieve a desired confidence interval. Of course, since standard deviations vary a great deal between studies, any particular usability test might well have a higher or lower standard deviation than the values given here, and one should perform further statistical tests on the actual data measured in the study.

Anyway, the results show that error rates tend to have the highest variability, meaning that they will generally require more test users to achieve the same level of confidence as can be achieved with fewer test users for measures of learnability and even fewer for measures of expert user performance. Figure 17 shows the confidence intervals for several possible desired levels of confidence from 50% to 90% in steps of 10%, as well as for 95%, assuming that the underlying usability measures follow a normal distribution. A confidence level of 95% is often used for research studies, but for practical development purposes, it may be enough to aim for an 80% level of confidence (the third curve from the top in each of the three nomographs).

The values on the y-axis should be interpreted as follows: The confidence interval (corresponding to the confidence level of one of the curves) is plus or minus that many percent of the measured mean value of the usability attribute. For example, assume that we are interested in measuring expert-user performance well enough that there is a 90% chance that the true value is no more than 15%

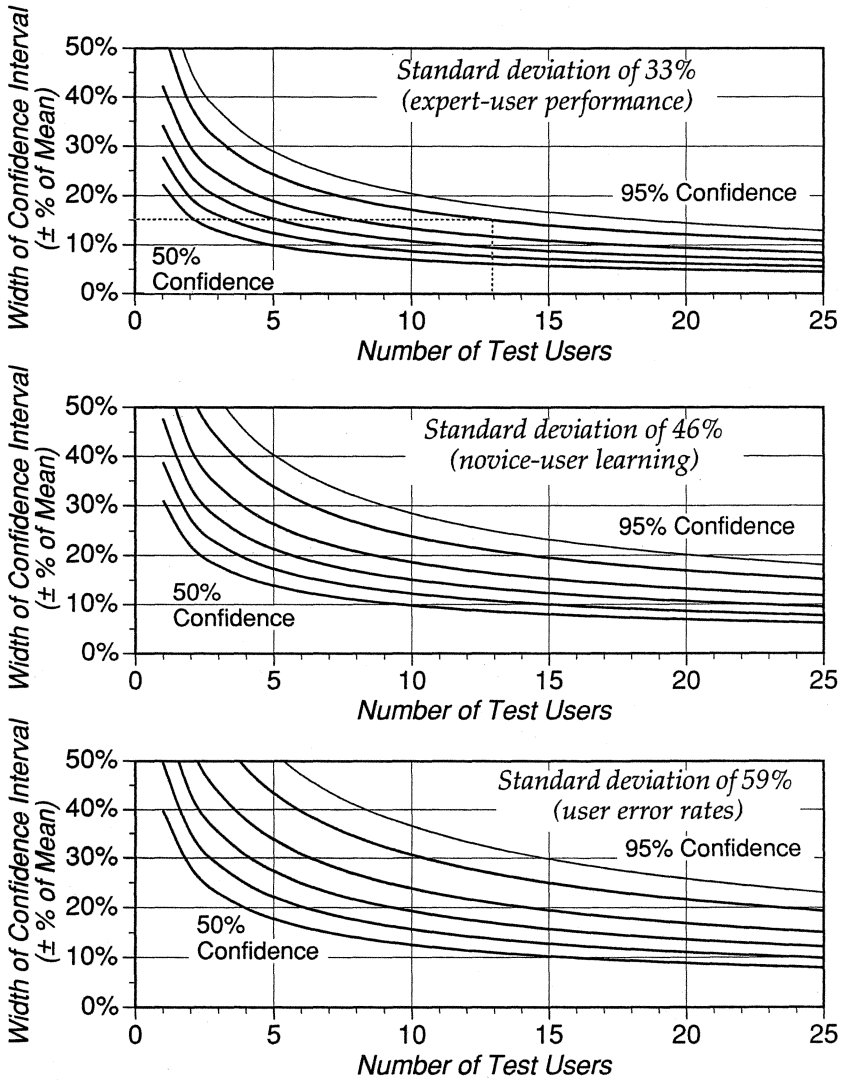

Figure 17 *Confidence intervals for usability testing from 1 to 25 users. Graphs for three different typical levels of standard deviations of the mean measured value. In each graph, the bottom curve is the 50% confidence level, followed by curves for 60%, 70%, 80%, 90%, and 95% (the top curve). The stippled lines in the top graph are discussed in the text.*

different from the mean value measured in our usability test. To find the necessary number of test users, we would start at the 15% mark on the *y*-axis in the top diagram in Figure 17 (corresponding to the desired length of the interval) and find the corresponding point on the 90% confidence level curve. We would then drop a line from that point to the *x*-axis to find the necessary number of test users (about 13). The diagram also shows that 5 test users would give us only a 70% probability of getting within ±15% of the true mean and that our 90% confidence interval would be ±24%. This level of accuracy might be enough for many projects.

Validity

Validity is a question of whether the usability test in fact measures something of relevance to usability of real products in real use outside the laboratory. Whereas reliability can be addressed with statistical tests, a high level of validity requires methodological understanding of the test method one is using as well as some common sense.

Typical validity problems involve using the wrong users or giving them the wrong tasks or not including time constraints and social influences. For example, a management information system might be tested with business school students as test users, but it is likely that the results would have been somewhat different if it had been tested with real managers. Even so, at least the business school students are people who likely will *become* managers, so they are probably more valid test users than, say, chemistry students. Similarly, results from testing a hypertext system with a toy task involving a few pages of text may not always be relevant for the use of the system in an application with hundreds of megabytes of information.

Confounding effects may also lower the validity of a usability test. For example, assume that you want to investigate whether it would be worthwhile to move from a character-based user interface to a graphical user interface for a certain application. You test this by comparing two versions of the system: one running on a 24×80 alphanumeric screen and one running on a 1024×1024 pixel graphics display. At a first glance, this may seem a reasonable

test to answer the question, but more careful consideration shows that the comparison between the two screens is as much a comparison between large and small screens as it is between character-based and graphical user interfaces.

6.1 Test Goals and Test Plans

Before any testing is conducted, one should clarify the purpose of the test since it will have significant impact on the kind of testing to be done. A major distinction is whether the test is intended as a formative or summative evaluation of the user interface. *Formative evaluation* is done in order to help improve the interface as part of an iterative design process. The main goal of formative evaluation is thus to learn which detailed aspects of the interface are good and bad, and how the design can be improved. A typical method to use for formative evaluation is a thinking-aloud test. In contrast, *summative evaluation* aims at assessing the overall quality of an interface, for example, for use in deciding between two alternatives or as a part of competitive analysis to learn how good the competition really is.[3] A typical method to use for summative evaluation is a measurement test.

Test Plans

A test plan should be written down before the start of the test and should address the following issues:

- The goal of the test: What do you want to achieve?
- Where and when will the test take place?
- How long is each test session expected to take?
- What computer support will be needed for the test?
- What software needs to be ready for the test?
- What should the state of the system be at the start of the test?

3. Remember, by the way, that manual or paper-based solutions that do not involve computers at all are also in the running and should be studied as well.

- What should the system/network load and response times be? If possible, the system should not be unrealistically slow (see the discussion of prototyping in Section 4.8), but neither should it be unrealistically fast because it is run on a system or network with no other users. One may have to artificially slow down the system to simulate realistic response times.
- Who will serve as experimenters for the test?
- Who are the test users going to be, and how are you going to get hold of them?
- How many test users are needed?
- What test tasks will the users be asked to perform?
- What criteria will be used to determine when the users have finished each of the test tasks correctly?
- What user aids (manuals, online help, etc.) will be made available to the test users?
- To what extent will the experimenter be allowed to help the users during the test?
- What data is going to be collected, and how will it be analyzed once it has been collected?
- What will the criterion be for pronouncing the interface a success? Often, this will be the "planned" level for the previously specified usability goals (see page 80), but it could also be a looser criterion such as "no new usability problems found with severity higher than 3."

Test Budget

The test plan should also include a budget for the test. Some costs will be out-of-pocket, meaning that they have to be paid cash. Other costs are in the nature of using company staff and resources that are already paid for. Such indirect costs may or may not be formally charged to the usability budget for the specific project, depending on how the company's accounting mechanisms are set up, but they should be included in the usability manager's internal budget for the test in any case. Typical cost elements of a user test budget are

- Usability specialists to plan, run, and analyze the test: out-of-pocket expense if consultants are used

- Administrative assistants to schedule test users, enter data, etc.
- Software developers to modify the code to include data collection or other desired test customization
- The test users' time: out-of-pocket expense if outside people are hired for the test
- Computers used during testing and during analysis
- The usability laboratory or other room used for the test
- Videotapes and other consumables: out-of-pocket expense

The cost estimates for the various staff members should be based on their loaded salary and not on their nominal salary. A loaded salary is the total cost to the company of having a person employed and includes elements like benefits, vacation pay, employment taxes or fees, and general corporate overhead.

The test budget should be split into fixed and variable costs, where fixed costs are those required to plan and set up the test no matter how many test users are run, and variable costs are the additional costs needed for each test user. Splitting the cost estimates in this way allows for better planning of the number of test users to include for each test. Obviously, both fixed and variable costs vary immensely between projects, depending on multiple factors such as the size of the interface and the salary level of the intended users. Based on several published budgets, estimates for a representative, medium-sized usability test can be derived, with fixed costs of $3,000 and variable costs of $1,000 per test user [Nielsen and Landauer 1993]. Note that any specific project is likely to have different costs than these estimates.

Given estimates for fixed and variable costs, it then becomes possible to calculate the optimal number of test users if further assumptions are made about the financial impact of finding usability problems and the probability of finding each problem with a single test user. Unfortunately, these latter two numbers are much harder to estimate than the costs of testing, but any given organization should be able to build up a database of typical values over time. Again based on values from published studies, a representative value of finding a usability problem in a medium-sized project can be taken as $15,000.

Nielsen and Landauer [1993] showed that the following formula gives a good approximation of the finding of usability problems:

$$\text{Usability\_Problems\_Found}(i) = N(1 - (1 - \lambda)^i),$$

where i is the number of test users, N is the total number of usability problems in the interface, and λ is the probability for finding any single problem with any single test user. The values of N and λ vary considerably between projects and should be estimated by curve-fitting as data becomes available for each project. It is also recommended that you keep track of these numbers for your own projects such that you can estimate "common" values of these parameters for use in the planning of future tests.

For several projects we studied, the mean number of problems in the interface, N, was 41 and the mean probability for finding any problem with a single user, λ, was 31% [Nielsen and Landauer 1993]. The following discussion uses these mean values to illustrate the use of the mathematical model in the budgeting of usability activities. Of course, one should really use the particular N and λ values that have been measured or estimated for the particular project one wants to analyze.

Given the assumptions mentioned above, Figure 18 shows how the pay-off ratio between the benefits and the costs changed in our average example with various numbers of test users. The highest ratio was achieved with three test users, where the projected benefits were $413,000, and the costs were $6,000. However, in principle, one should keep testing as long as the benefit from one additional test user is greater than the cost of running that user. Under the above model, this would imply running 15 test users at a cost of $18,000 to get benefits worth $613,000. If the recommendation in Section 4.10 to use iterative design is followed, it will be better to conduct more, smaller, tests (one for each iteration) than to spend everything on a single test.

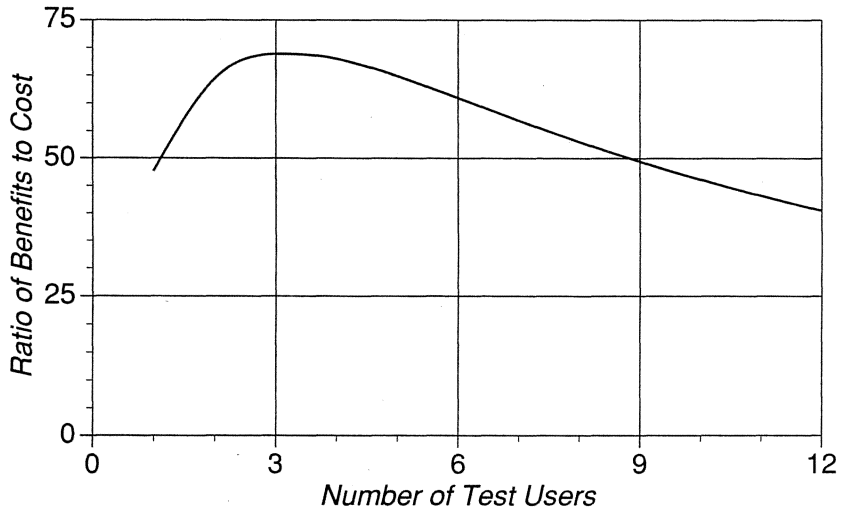

Figure 18 *The pay-off ratio (how much larger the benefits are than the costs) for user tests with various numbers of test users under the assumptions for a "typical" medium-sized project described in the text.*

Pilot Tests

No usability testing should be performed without first having tried out the test procedure on a few pilot subjects. Often, one or two pilot subjects will be enough, but more may be needed for large tests or when the initial pilot tests show severe deficiencies in the test plan. The first few pilot subjects may be chosen for convenience among people who are easily available to the experimenter even if they are not representative of the actual users, since some mistakes in the experimental design can be found even with subjects such as one's colleagues. Even so, at least one pilot subject should be taken from the same pool as the other test users.

During pilot testing, one will typically find that the instructions for some of the test tasks are incomprehensible to the users or that they misinterpret them. Similarly, any questionnaires used for subjective satisfaction rating or other debriefing will often need to be changed based on pilot testing. Also, one very often finds a mismatch between the test tasks and the time planned for each test

session. Most commonly, the tasks are more difficult than one expected, but of course it may also be the case that some tasks are too easy. Depending on the circumstances of the individual project, one will either have to revise the tasks or make more time available for each test session.

Pilot testing can also be used to refine the experimental procedure and to clarify the definitions of various things that are to be measured. For example, it is often difficult to decide exactly what constitutes a user error or exactly when the user can be said to have completed a given test task, and the pilot test may reveal inconsistencies or weaknesses in the definitions contained in the test plan.

6.2 Getting Test Users

The main rule regarding test users is that they should be as representative as possible of the intended users of the system. If the test plan calls for a "discount usability" approach with very few test users, one should not choose users from outlier groups but should take additional care to involve average users. If more test users are to be used, one should select users from several different subpopulations to cover the main different categories of expected users.

The main exception from the rule that test users should be representative of the end users is testing with sales people. For many products, sales staff is used to give demonstrations to prospective customers, and the ease with which they can give these demos may be a major selling point. Often, sales people handle multiple products and do not get extensive experience from actual use of any individual product. The experience of demonstrating a user interface is very different from that of actually using it for a real task, and even though most usability efforts should aim at making the system easier to use for the users, it may increase sales significantly if "demoability" is also considered as a usability attribute.

Sometimes, the exact individuals who will be using a system can be identified. This is typically the case for systems that are being developed internally in a company for use in a given department

of that company. In this case, representative users are easy to find, even though it may present some difficulties to get them to spend time on user testing instead of doing their primary job. Internal test users are often recruited through the users' management who agrees to provide a certain number of people. Unfortunately, managers often tend to select their most able staff members for such tests (either to make their department look good or because these staff members have the most interest in new technology), so one should explicitly ask managers to choose a broad sample with respect to salient user characteristics such as experience and seniority.

In other cases, the system is targeted at a certain type of users, such as lawyers, the secretaries in a dental clinic, or warehouse managers in small manufacturing companies. These user groups can be more or less homogeneous, and it may be desirable to involve test users from several different customer locations. Sometimes, existing customers are willing to help out with the test since it will get them an early look at new software as well as improving the quality of the resulting product, which they will be using. In other cases, no existing customers are available, and it may be more difficult to gain access to representative users. Sometimes, test users can be recruited from temporary employment agencies, or it may be possible to get students in the domain of interest from a local university or trade school. It may also be possible to enter a classified advertisement under job openings in order to recruit users who are currently unemployed. Of course, it will be necessary to pay all users recruited with these latter methods.

Yet other software is intended for the general population, and one can in principle use anybody as test users, again using employment agencies, students, or classified advertising to recruit test users. Especially when testing students, one should consider whether the system is also intended to be used by older users, since they may have somewhat different characteristics [Czaja *et al.* 1989; Nielsen and Schaefer 1993] and may therefore need to be included as an additional test group. A good source of older test users is retirees, who may also serve as a pool of talent with specific domain expertise.

Novice versus Expert Users

One of the main distinctions between categories of users is that between novice and expert users (see also Section 2.5, *Categories of Users and Individual User Differences*, on page 43 for further dimensions of interest). Almost all user interfaces need to be tested with novice users, and many systems should also be tested with expert users. Typically, these two groups should be tested in separate tests with some of the same and some different test tasks.

Sometimes, one will have to train the users with respect to those aspects of a user interface that are unfamiliar to them but are not relevant for the main usability test. This is typically necessary during the transition from one interface generation to the next, where users will have experience with the old interaction techniques but will be completely baffled by the new ones unless they are given some training. For example, a company that is moving from character-based interfaces to graphical user interfaces will have many users who have never used a mouse before, and these users will have to be trained in the use of the mouse before it is relevant to use them as test users of a mouse-based system. Using a mouse is known to be hard for the first several hours, and it is almost impossible to use a mouse correctly the first few minutes. If users are not trained in the use of the mouse and other standard interaction techniques before they are asked to test a new interface, the test will be completely dominated by the effects of the users' struggle with the interaction devices and techniques, and no information will be gained as to the usability of the dialogue.

One example of the potentially devastating effect of not training users before a test was a study of the use of a single window versus multiple windows for an electronic book [Tombaugh *et al.* 1987]. When novice users without any specific training were used as test subjects, the single-window interface was best for reading the electronic book. The time needed to answer questions about the text was 85 seconds when using the multiwindow interface and 72 seconds when using the single-window interface. In a second test, the test users were first given 30 minutes' training in the use of a mouse to control multiple windows, and the time to answer the questions about the text was now 66 seconds for the single-

window interface and only 56 seconds for the multiwindow interface. Thus, both interfaces benefited from having more experienced users, but the most interesting result is that the overall conclusion with respect to determining the "winner" of the test came out the opposite. The single-window solution would be best for a "walk-up-and-use" system for users without prior mouse experience. On the other hand, the multiwindow solution would be best in the more common case where the electronic book was to be used in an office or school environment where people would be using the same computer extensively. For such environments, the wrong conclusion would have been drawn if a test with untrained users had been the only one.

Between-Subjects versus Within-Subjects Testing

Often, usability testing is conducted in order to compare the usability of two or more systems. If so, there are two basic ways of employing the test users: *between-subject testing* and *within-subject testing*.

Between-subject testing is in some sense the simplest and most valid since it involves using different test users for the different systems. Thus, each test user only participates in a single test session. The problem with between-subject designs is the huge individual variation in user skills referred to on page 166. Therefore, it can be necessary to run a very large number of test users in each condition in order to smooth over random differences between test users in each of the groups.

Between-subject testing also risks a bias due to the assignment of test users to the various groups. For example, one might decide to test 20 users, call for volunteers, and assign the first 10 users to one system and the next 10 to the other. Even though this approach may seem reasonable, it in fact introduces a bias, since users who volunteer early are likely to be different from users who volunteer late. For example, early volunteers may be more conscientious in reading announcements, or they may be more interested in new technology, and thus more likely to be super-users. There are two methodologically sound ways to assign test users to groups: The simplest and best is random assignment, which minimizes the risk

of any bias but requires a large number of test users because of individual variability. The second method is matched assignment, which involves making sure that each group has equally many users from each of those categories that have been defined as being of interest to the test. For example, users from different departments might be considered different categories, as may old versus young users, men versus women, and users with different computer experience or different educational backgrounds.

Alternatively, one may conduct the test as a within-subject design, meaning that all the test users get to use all the systems that are being tested. This method automatically controls for individual variability since any user who is particularly fast or talented will presumably be about equally superior in each test condition. Within-subject testing does have the major disadvantage that the test users cannot be considered as novice users anymore when they approach the other systems after having learned how to use the first system. Often, some transfer of skill will take place between systems, and the users will be better at using the second system than they were at using the first. In order to control for this effect, users are normally divided into groups, with one group using one system first and the other group using the other system first. The issues discussed above regarding the assignment of users to groups also apply to this aspect of within-subject testing.

6.3 *Choosing Experimenters*

No matter what test method is chosen, somebody has to serve as the experimenter and be in charge of running the test. In general, it is of course preferable to use good experimenters who have previous experience in using whatever method is chosen. For example, a study where 20 different groups of experimenters tested the same interface, there was a correlation of $r = .76$ between the rated quality of the methodology used by a group and the number of usability problems they discovered in the interface [Nielsen 1992a]. When running 3 test subjects, experimenters using very

good methodology found about 5–6 of the 8 usability problems in the interface, and experimenters using very poor methodology only found about 2–3 of the problems.

This result does not mean that one should abandon user testing if no experienced usability specialist is available to serve as the experimenter. First, it is obviously better to find a few usability problems than not to find any, and second, even inexperienced experimenters can use a decent (if not perfect) methodology if they are careful. It is possible for computer scientists to learn user test methods and apply them with good results [Nielsen 1992a; Wright and Monk 1991].

In addition to knowledge of the test method, the experimenter must have extensive knowledge of the application and its user interface. System knowledge is necessary for the experimenter to understand what the users are doing as they perform tasks with the system, and to make reasonable inferences about the users' probable intentions at various stages of the dialogue. Often, users' actions will go by too fast for experimenters, who are trying to understand what the system is doing at the same time as they are analyzing the users.

The experimenter does not necessarily need to know how the system is implemented, even though such knowledge can come in handy during tests of preliminary prototypes with a tendency to crash. If the experimenter does not know how to handle system crashes, it is a good idea to arrange to have a programmer with the necessary skills stand by in a nearby office.

One way to get experimenters with a high degree of system knowledge is to use the system's designers themselves as evaluators [Wright and Monk 1991]. In addition to the practical advantages, there are also motivational reasons for doing so, since the experience of seeing users struggle with their system always has a very powerful impact on designers [Jørgensen 1989]. There are some problems with having people run tests of their own systems, though, including a possible lack of objectivity that may lead them to help the users too much (see also the footnote on page 204). A

common weakness is the tendency for a designer to explain away user problems rather than acknowledging them as real issues. To avoid these problems, developers can serve as one *part* of the usability testing team while usability specialists handle relations with the users [Ehrlich *et al.* 1994].

6.4 *Ethical Aspects of Tests with Human Subjects*

Users are human, too. Therefore, one cannot subject them to the kind of "destructive testing" that is popular in the components industry. Instead, tests should be conducted with deep respect for the users' emotions and well-being [Allen 1984; American Psychological Association 1982].

At first, it might seem that usability testing does not represent the same potential dangers to the users as would, say, participation in a test of a new drug. Even though it is true that usability test subjects are not normally bodily harmed, even by irate developers resenting the users' mistreatment of their beloved software, test participation can still be quite a distressful experience for the users [Schrier 1992]. Users feel a tremendous pressure to perform, even when they are told that the purpose of the study is to test the system and not the user. Also, users will inevitably make errors and be slow at learning the system (especially during tests of early designs with many severe usability problems), and they can easily get to feel inadequate or stupid as they experience these difficulties. Knowing that they are observed, and possibly recorded, makes the feeling of performing inadequately even more unpleasant to the users. Test users have been known to break down and cry during usability testing, even though this only happens in a small minority of cases.

At first, one might think that highly educated and intelligent users would have enough self-confidence to make fear of inadequacy

less of a problem. On the contrary, high-level managers and highly specialized professionals are often especially concerned about exhibiting ignorance during a test. Therefore, experimenters should be especially careful to acknowledge the professional skills of such users up front and emphasize the need to involve people with these users' particular knowledge in the test.

The experimenter has a responsibility to make the users feel as comfortable as possible during and after the test. Specifically, the experimenter must never laugh at the users or in any way indicate that they are slow at discovering how to operate the system. During the introduction to the test, the experimenter should make clear that it is the system that is being tested and not the user. To emphasize this point, test users should never be referred to as "subjects," "guinea pigs," or other such terms. I personally prefer the term "test user," but some usability specialists like to use terms such as "usability evaluator" or "participant," which emphasize even more that it is the system that is being tested. Since the term "evaluator" technically speaking refers to an inspection-oriented role where usability specialists judge a system instead of *using* it to perform a task, I normally do not use this term myself when referring to test users.

The users should be told that no information about the performance of any individual users will be revealed and specifically that their manager will not be informed about their performance. The test itself should be conducted in a relaxed atmosphere, and the experimenter should take the necessary time for small talk to calm down the user before the start of the experiment, as well as during any breaks. It might also be a good idea to serve coffee, soft drinks, or other refreshments—especially if the test takes more than an hour or so. Furthermore, to bolster the users' confidence and make them at ease, the very first test task should be so easy that they are virtually guaranteed an early success experience.

The experimenter should ensure that the test room, test computer, and test software are ready before the test user arrives in order to avoid the confusion that would otherwise arise due to last-minute adjustments. Also, of course, copies of the test tasks, any questionnaires, and other test materials should be checked before the arrival of the user such that they are ready to be handed out at the appropriate time. The test session should be conducted without disruptions: typically, one should place a sign saying, *"User test in progress—Do not disturb"* outside the (closed) door and disable any telephone sets in the test room.

The test results should be kept confidential, and reports from the test should be written in such a way that individual test users cannot be identified. For example, users can be referred to by numbers (User1, User2, etc.) and not by names or even initials.[4] The test should be conducted with as few observers as possible, since the size of the "audience" also has a detrimental effect on the test user: It is less embarrassing to make a fool of yourself in front of 1 person than in front of 10. And remember that users *will* think that they are making fools of themselves as they struggle with the interface and overlook "obvious" options, even if they only make the same mistakes as everybody else. For similar reasons, videotapes of a user test session should not be shown publicly without explicit permission from the user. Also, the users' manager should never be allowed to observe the test for any reason and should not be given performance data for individual users.

During testing, the experimenter should normally not interfere with the user but should let the user discover the solutions to the problems on his or her own. Not only does this lead to more valid and interesting test results,[5] it also prevents the users from feeling that they are so stupid that the experimenter had to solve the problems for them. On the other hand, the experimenter should not let a user struggle endlessly with a task if the user is clearly bogged down and getting desperate. In such cases, the experimenter can

4. Ensuring anonymity requires a fair amount of thought. For example, a report of a test with users drawn from a department with only one female staff member referred to all users as "he," even when describing observations of the woman since her anonymity would otherwise have been compromised.

5. It is a common mistake to help users too early. Since users normally do not get help when they have to learn a computer system on their own, there is highly relevant information to be gained from seeing what further difficulties users get into as they try to solve the problem on their own.

Before the test:
Have everything ready before the user shows up.
Emphasize that it is the *system* that is being tested, not the user.
Acknowledge that the software is new and untested, and may have problems.
Let users know that they can stop at any time.
Explain any recording, keystroke logging, or other monitoring that is used.
Tell the user that the test results will be kept completely confidential.
Make sure that you have answered all the user's questions before proceeding.

During the test:
Try to give the user an early success experience.
Hand out the test tasks one at a time.
Keep a relaxed atmosphere in the test room, serve coffee and/or have breaks.
Avoid disruptions: Close the door and post a sign on it. Disable telephone.
Never indicate in any way that the user is making mistakes or is too slow.
Minimize the number of observers at the test.
Do not allow the user's management to observe the test.
If necessary, have the experimenter stop the test if it becomes too unpleasant.

After the test:
End by stating that the user has helped you find areas of improvement.
Never report results in such a way that individual users can be identified.
Only show videotapes outside the usability group with the user's permission.

Table 9 *Main ethical considerations for user testing.*

gently provide a hint or two to the user in order to get on with the test. Also, the experimenter may have to terminate the test if the user is clearly unhappy and unable to do anything with the system. Such action should be reserved for the most desperate cases only. Furthermore, test users should be informed before the start of the test that they can always stop the test at any time, and any such requests should obviously be honored.

After the test, the user should be debriefed and allowed to make comments about the system. After the administration of the questionnaire (if used), any deception employed in the experiment should be disclosed in order not to have the user leave the test with an erroneous understanding of the system. An example of a deception that should be disclosed is the use of the Wizard of Oz method (see page 96) to simulate nonexisting computer capabilities. Also, the experimenter can answer any additional user questions that

could not be answered for fear of causing bias until after the user had filled in the questionnaire. The experimenter should end the debriefing by thanking the user for participating in the test and explicitly state that the test helped to identify areas of possible improvement in the product.[6] This part of the debriefing helps users recover their self-respect after the many errors they probably felt they made during the test itself. Also, the experimenter should endeavor to end the session on a positive and relaxed note, repeating that the results are going to be kept confidential and also engaging in some general conversation and small talk as the user is being escorted out of the building or laboratory area.

Table 9 summarizes the most important ethical considerations for user testing. In addition to following the rules outlined here, it is a good idea for the experimenters to have tried the role of test subjects themselves a few times, so that they know from personal experience how stupid and vulnerable subjects may feel.

6.5 *Test Tasks*

The basic rule for test tasks is that they should be chosen to be as representative as possible of the uses to which the system will eventually be put in the field. Also, the tasks should provide reasonable coverage of the most important parts of the user interface. The test tasks can be designed based on a task analysis or based on a product identity statement listing the intended uses for the product. Information from logging frequencies of use of commands in running systems (see page 217) and other ways of learning how users actually use systems, such as field observation, can also be used to construct more representative sets of test tasks for user testing of similar systems [Gaylin 1986].

6. However, it may also be necessary to mention that the development team will not necessarily be able to correct all identified problems. Users can get very disappointed if they find that the system is released with one of "their" problems still in the interface in spite of a promise to correct it.

The tasks need to be small enough to be completed within the time limits of the user test, but they should not be so small that they become trivial. The test tasks should specify precisely what result the user is being asked to produce, since the process of using a computer to achieve a goal is considerably different from just playing around. For example, a test task for a spreadsheet could be to enter sales figures for six regions for each of four quarters, with some sample numbers given in the task description. A second test task could then be to obtain totals and percentages, and a third might be to construct a bar chart showing trends across regions. Test tasks should normally be given to the users in writing. Not only does this ensure that all users get the tasks described the same way, but having written tasks also allows the user to refer to the task description during the experiment instead of having to remember all the details of the task. After the user has been given the task and has had a chance to read it, the experimenter should allow the user to ask questions about the task description, in order to minimize the risk that the user has misinterpreted the task. Normally, task descriptions are handed to the user on a piece of paper, but they can also be shown in a window on the computer. This latter approach is usually chosen in computer-paced tests where users have to perform a very large number of tasks.

Test tasks should never be frivolous, humorous, or offensive, such as testing a paint program by asking the user to draw a mustache on a scanned photo of the President. First, there is no guarantee that everybody will find the same thing funny, and second, the nonserious nature of such tasks distracts from the test of the system and may even demean the users. Instead, all test tasks should be business-oriented (except, of course, for tests of entertainment software and such) and as realistic as possible. To increase both the users' understanding of the tasks and their sense of being realistic usage of the software, the tasks can be related to an overall scenario. For example, the scenario for the spreadsheet example mentioned above could be that the user had just been hired as sales manager for a company and had been asked to give a presentation the next day.

The test tasks can also be used to increase the user's confidence. The very first test task should always be extremely simple in order to guarantee the user an early success experience to boost morale. Similarly, the last test task should be designed to make users feel that they have accomplished something. For example, a test of a word processor could end with having the user print out a document. Since users will feel inadequate if they do not complete all the given tasks, one should never give the users a complete listing of all the test tasks in advance. Rather, the tasks should be given to the users one at a time such that it is always possible to stop the test without letting the user feel incompetent.

6.6 *Stages of a Test*

A usability test typically has four stages:

1. Preparation
2. Introduction
3. The test itself
4. Debriefing

Preparation

In preparation for the experiment, the experimenter should make sure that the test room is ready for the experiment, that the computer system is in the start state that was specified in the test plan, and that all test materials, instructions, and questionnaires are available. For example, all files needed for the test tasks should be restored to their original content, and any files created during earlier tests should be moved to another computer or at least another directory. In order to minimize the user's discomfort and confusion, this preparation should be completed before the arrival of the user. Also, any screen savers should be switched off, as should any other system components, such as email notifiers, that might otherwise interrupt the experiment.

Introduction

During the introduction, the experimenter welcomes the test user and gives a brief explanation of the purpose of the test. The experimenter may also explain the computer setup to users if it is likely to be unfamiliar to them. The experimenter then proceeds with introducing the test procedure. Especially for inexperienced experimenters, it may be a good idea to have a checklist at hand with the most important points to be covered, but care should be taken not to make the introduction seem mechanical, as could easily be the case if the experimenter were to simply read from the checklist.

Typical elements to cover in a test introduction include the following:

- The purpose of the test is to evaluate the software and not the user.
- Unless the experimenter is actually the system designer, the experimenter should mention that he or she has no personal stake in the system being evaluated, so that the test user can speak freely without being afraid of hurting the experimenter's feelings. If the experimenter *did* design the system, this fact is probably better left unsaid in order to avoid the opposite effect.
- The test results will be used to improve the user interface, so the system that will eventually be released will likely be different from the one seen in the test.
- A reminder that the system is confidential and should not be discussed with others. Even if the system is not confidential, it may still be a good idea to ask the test user to refrain from discussing it with colleagues who may be participating in future tests, in order not to bias them.
- A statement that participation in the test is voluntary and that the user may stop at any time.[7]

7. Even though this may not need to be mentioned explicitly in the experimenter's introduction, users who do elect to stop the experiment should still get whatever payment was promised for the time they have spent, even if they did not complete the experiment, and even if the data from their participation cannot be used.

- A reassurance that the results of the test will be kept confidential and not shown to anybody in a form where the individual test user can be identified.
- An explanation of any video or audio recording that may be taking place. In cases where the video record will not be showing the user's face anyway, but only the screen and keyboard and the user's back, it is a good idea to mention this explicitly to alleviate the user's worries about being recorded.
- An explanation that the user is welcome to ask questions since we want to know what they find unclear in the interface, but that the experimenter will not answer most questions during the test itself, since the goal of the test is to see whether the system can be used without outside help.
- Any specific instructions for the kind of experiment that is being conducted, such as instructions to think out loud or to work as fast as possible while minimizing mistakes.
- An invitation to the user to ask any clarifying questions before the start of the experiment.

Many people have the test users sign an informed consent form that repeats the most important instructions and experimental conditions and states that they have understood them. I do not like these forms since they can increase the user's anxiety level by making the test seem more foreboding than it really is. Sometimes, consent forms may be required for legal reasons, and they should certainly be used in cases where videotapes or other records or results from the test will be shown to others. In any case, it is recommended to keep any such forms short, to the point, and written in everyday language rather than legalese, so that the users do not fear that they are being entrapped to sign away more rights than they actually are.

During the introduction phase, the experimenter should also ensure that the physical set-up of the computer is ergonomically suited for the individual test user. A common problem is the position of the mouse for left-handed users, but it may also be necessary to adjust the chair or other parts of the room such that the user feels comfortable. If the actual computer model is unfamiliar to the user, it may be a good idea to let the user practice using some other

software before the start of the test itself, to avoid contaminating the test results with the user's initial adjustments to the hardware.

After the introduction, the user is given any written instructions for the test, including the first test task, and asked to read them. The experimenter should explicitly ask the test user whether he or she has any questions regarding the experimental procedure, the test instructions, or the tasks before the start of the test.

Running the Test

During the test itself, the experimenter should normally refrain from interacting with the user, and should certainly not express any personal opinions or indicate whether the user is doing well or poorly. The experimenter may make uncommitted sounds like "uh-huh" to acknowledge comments from the user and to keep the user going, but again, care should be taken not to let the tone of such sounds indicate whether the user is on the right track or has just made a ridiculous comment. Also, the experimenter should refrain from helping the test user, even if the user gets into quite severe difficulties.

The main exception from the rule that users should not be helped is when the user is clearly stuck and is getting unhappy with the situation. The experimenter may also decide to help a user who is encountering a problem that has been observed several times before with previous test users. The experimenter should only do so if it is clear beyond any doubt from the previous tests what the problem is and what different kinds of subsequent problems users may encounter as a result of the problem in question. It is tempting to help too early and too much, so experimenters should exercise caution in deciding when to help. Also, of course, no help can be given during experiments aiming to time users' performance on a task.

In case several people are observing the experiment, it is important to have appointed one of them as the official experimenter in advance and only have that one person provide instructions and speak during the experiment. In order not to confuse the user, all other observers should keep completely quiet, even if they do not

agree with the way the experimenter is running the experiment. If they absolutely need to make comments, they can do so by unobtrusively passing the experimenter a note or talking with the experimenter during a break.

Debriefing

After the test, the user is debriefed and is asked to fill in any subjective satisfaction questionnaires. In order to avoid any bias from comments by the experimenter, questionnaires should be administered *before* any other discussion of the system. During debriefing, users are asked for any comments they might have about the system and for any suggestions they may have for improvement. Such suggestions may not always lead to specific design changes, and one will often find that different users make completely contradictory suggestions, but this type of user suggestion can serve as a rich source of additional ideas to consider in the redesign.

The experimenter can also use the debriefing to ask users for further comments about events during the test that were hard for the experimenter to understand. Even though users may not always remember why they did certain things, they are sometimes able to clarify some of their presumptions and goals.

Finally, as soon as possible after the user has left, the experimenter should check that all results from the test have been labelled with the test user's number, including any files recorded by the computer, all questionnaires and other forms, as well as the experimenter's own notes. Also, the experimenter should write up a brief report on the experiment as soon as possible, while the events are still fresh in the experimenter's mind and the notes still make sense. A full report on the complete sequence of experiments may be written later, but the work of writing such a report is made considerably simpler by having well-organized notes and preliminary reports from the individual tests.

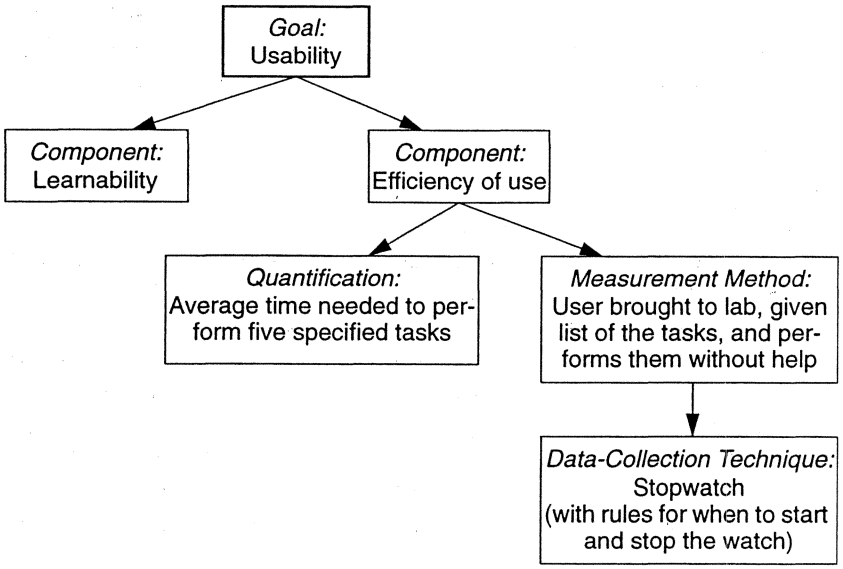

Figure 19 *Model of usability measurement*

6.7 Performance Measurement

Measurement studies form the basis of much traditional research on human factors and are also important in the usability engineering lifecycle for assessing whether usability goals have been met (see page 79) and for comparing competing products. User performance is almost always measured by having a group of test users perform a predefined set of test tasks while collecting time and error data.

A major pitfall with respect to measurement is the potential for measuring something that is poorly related to the property one is really interested in assessing. Figure 19 shows a simple model relating the true goal of a measurement study (the usability of the system) to the actual data-collection activities that may sometimes erroneously be thought of as the core of measurement. As indicated by the model, one starts out by making clear the goal of the exercise. Here, we will assume that "usability" as an abstract concept is

the goal, but it could also be, e.g., improved customer perceptions of the quality of a company's user interfaces.

Goals are typically quite abstract, so one then breaks them down into components like the usability attributes discussed further in Section 2.2. Figure 19 shows two such components, learnability and efficiency of use. As further discussed in Section 4.3, one then needs to balance the various components of the goal and decide on their relative importance. Once the components of the goal have been defined, it becomes necessary to quantify them precisely. For example, the component "efficiency of use" can be quantified as the average time it takes users to perform a certain number of specified tasks. Even if these tasks are chosen to be representative of the users' normal task mix, it is important to keep in mind that the test tasks are only that: test tasks and not all possible tasks. In interpreting the results from the measurement study, it is necessary to keep in mind the difference between the principled component that one is aiming for, that is, efficiency of use in general, and the specific quantification which is used as a proxy for that component (i.e., the test tasks). As an obvious example, an iterative design process should not aim at improving efficiency of use for a system just by optimizing the interface for the execution of the five test tasks and nothing else (unless the tasks truly represent all of what the user ever will do with the system).

Given the quantification of a component, one needs to define a method for measuring the users' performance. Two obvious alternatives come to mind for the example in Figure 19: either bring some test users into the laboratory and give them a list of the test tasks to perform, or observe a group of users at work in their own environment and measure them whenever a task like the specified test tasks occurs. Finally, one needs to define the actual activities that are to be carried out to collect the data from the study. Some alternatives for the present example could be to have the computer measure the time from start to end of each task, to have an experimenter measure it by a stopwatch, and to have users report the time themselves in a diary. In either case it is important to have a clear definition of when a task starts and when it stops.

Typical quantifiable usability measurements include

- The time users take to complete a specific task.
- The number of tasks (or the proportion of a larger task) of various kinds that can be completed within a given time limit.
- The ratio between successful interactions and errors.
- The time spent recovering from errors.
- The number of user errors.
- The number of immediately subsequent erroneous actions.
- The number of commands or other features that were utilized by the user (either the absolute number of commands issued or the number of *different* commands and features used).
- The number of commands or other features that were never used by the user.
- The number of system features the user can remember during a debriefing after the test.
- The frequency of use of the manuals and/or the help system, and the time spent using these system elements.
- How frequently the manual and/or help system solved the user's problem.
- The proportion of user statements during the test that were positive versus critical toward the system.
- The number of times the user expresses clear frustration (or clear joy).
- The proportion of users who say that they would prefer using the system over some specified competitor.
- The number of times the user had to work around an unsolvable problem.
- The proportion of users using efficient working strategies compared to the users who use inefficient strategies (in case there are multiple ways of performing the tasks).
- The amount of "dead" time when the user is not interacting with the system. The system can be instrumented to distinguish between two kinds of dead time: response-time delays where the user is waiting for the system, and thinking-time delays where the system is waiting for the user. These two kinds of dead time should obviously be approached in different ways.

- The number of times the user is sidetracked from focusing on the real task.

Of course, only a subset of these measurements would be collected during any particular measurement study.

6.8 *Thinking Aloud*

Thinking aloud may be the single most valuable usability engineering method. Basically, a thinking-aloud test involves having a test subject use the system while continuously thinking out loud [Lewis 1982]. By verbalizing their thoughts, the test users enable us to understand how they view the computer system, and this again makes it easy to identify the users' major misconceptions. One gets a very direct understanding of what parts of the dialogue cause the most problems, because the thinking-aloud method shows how users interpret each individual interface item.

The thinking-aloud method has traditionally been used as a psychological research method [Ericsson and Simon 1984], but it is increasingly being used for the practical evaluation of human–computer interfaces [Denning *et al.* 1990]. The main disadvantage of the method is that is does not lend itself very well to most types of performance measurement. On the contrary, its strength is the wealth of qualitative data it can collect from a fairly small number of users. Also, the users' comments often contain vivid and explicit quotes that can be used to make the test report more readable and memorable.

At the same time, thinking aloud may also give a false impression of the cause of usability problems if too much weight is given to the users' own "theories" of what caused trouble and what would help. For example, users may be observed to overlook a certain field in a dialog box during the first part of a test. After they finally find the field, they may claim that they would have seen it immediately if it had been in some other part of the dialog box. It is important not to rely on such statements. Instead, the experimenter should make notes of what the users were *doing* during the part of

the experiment where they overlooked the critical field. Data showing where users actually looked has much higher validity than the users' claim that they would have seen the field if it had been somewhere else. The strength of the thinking-aloud method is to show what the users are doing and why they are doing it *while* they are doing it in order to avoid later rationalizations.

Thinking out loud seems very unnatural to most people, and some test users have great difficulties in keeping up a steady stream of utterances as they use a system.[8] Not only can the unnaturalness of the thinking aloud situation make the test harder to conduct, but it can also impact the results. First, the need to verbalize can slow users down, thus making any performance measurements less representative of the users' regular working speed. Second, users' problem solving behavior can be influenced by the very fact that they are verbalizing their thoughts. The users might notice inconsistencies in their own models of the system, or they may concentrate more on critical task components [Bainbridge 1979], and these changes may cause them to learn some user interfaces faster or differently than they otherwise would have done. For example, Berry and Broadbent [1990] provided users with written instructions on how to perform a certain task and found that users performed 9% faster if they were asked to think aloud while doing the task. Berry and Broadbent argue that the verbalization reinforced those aspects of the instructions which the users needed for the task, thus helping them become more efficient. In another study [Wright and Converse 1992], users who were thinking aloud while performing various file system operations were found to make only about 20% of the errors made by users who were working silently. Furthermore, the users in the thinking-aloud study finished their tasks about twice as fast as the users in the silent condition.

8. Verbalization seems to come the hardest to expert users who may perform many operations so quickly that they have nothing to say. They may not even consciously know what they are doing in cases where they have completely automated certain common procedures.

The experimenter will often need to continuously prompt the user to think out loud by asking questions like, "What are you thinking now?" and "What do you think this message means?" (*after* the user has noticed the message and is clearly spending time looking at it and thinking about it). If the user asks a question like, "Can I do such-and-such?" the experimenter should not answer, but instead keep the user talking with a counter-question like, "What do you think will happen if you do it?" If the user acts surprised after a system action but does not otherwise say anything, the experimenter may prompt the user with a question like, "Is that what you expected would happen?" Of course, following the general principle of not interfering in the user's use of the system, the experimenter should *not* use prompts like, "What do you think the message on the bottom of the screen means?" if the user has not noticed that message yet.

Since thinking aloud seems strange to many people, it may help to give the test users a role model by letting them observe a short thinking-aloud test before the start of their own experiment. One possibility is for the experimenter to enact a small test, where the experimenter performs some everyday task like looking up a term in a dictionary while thinking out loud. Alternatively, users can be shown a short video of a test that was made with the sole purpose of instructing users. Showing users how a test videotape looks may also help alleviate their own fears of any videotaping that will be done during the test.

Users will often make comments regarding aspects of the user interface which they like or do not like. To some extent, it is one of the great advantages of the thinking-aloud method that one can collect such informal comments about small irritants that would not show up in other forms of testing. They may not impact measurable usability, but they might as well be fixed. Unfortunately, users will often disagree about such irritants, so one should take care not to change an interface just because of a comment by a single user. Also, user comments will often be inappropriate when seen in a larger interface design perspective, so it is the responsibility of the experimenter to interpret the user's comments and not just accept them indiscriminately. For example, users who are

using a mouse for the first time will often direct a large proportion of their comments toward aspects of moving the mouse and pointing and clicking, which might be interesting for a designer of more intuitive input hardware but are of limited use to a software designer. In such a test, the experimenter would need to abstract from the users' mouse problems and try to identify the underlying usability problems in the dialogue and estimate how the users would have used the interface if they had been better at using the pointing device.

Constructive Interaction

A variation of the thinking-aloud method is called *constructive interaction* and involves having two test users use a system together [O'Malley *et al.* 1984]. This method is sometimes also called *codiscovery learning* [Kennedy 1989]. The main advantage of constructive interaction is that the test situation is much more natural than standard thinking-aloud tests with single users, since people are used to verbalizing when they are trying to solve a problem together. Therefore, users may make more comments when engaged in constructive interaction than when simply thinking aloud for the benefit of an experimenter [Hackman and Biers 1992]. The method does have the disadvantage that the users may have different strategies for learning and using computers. Therefore, the test session may switch back and forth between disparate ways of using the interface, and one may also occasionally find that the two test users simply cannot work together.

Constructive interaction is especially suited for usability testing of user interfaces for children since it may be difficult to get them to follow the instructions for a standard thinking-aloud test.

Constructive interaction is most suited for projects where it is easy to get large numbers of users into the lab, and where these users are comparatively cheap, since it requires the use of twice as many test users as single-user thinking aloud.

Retrospective Testing

If a videotape has been made of a user test session, it becomes possible to collect additional information by having the user review the recording [Hewett and Scott 1987]. This method is sometimes called *retrospective testing*. The user's comments while reviewing the tape are sometimes more extensive than comments made under the (at least perceived) duress of working on the test task, and it is of course possible for the experimenter to stop the tape and question the user in more detail without fearing to interfere with the test, which has essentially already been completed.

Retrospective testing is especially valuable in cases where representative test users are difficult to get hold of, since it becomes possible to gain more information from each test user. The obvious downside is that each test takes at least two times as long, so the method is not suited if the users are highly paid or perform critical work from which they cannot be spared for long. Unfortunately, those users who are difficult to get to participate in user testing are often exactly those who are also very expensive, but there are still some cases where retrospective testing is beneficial.

Coaching Method

The coaching method [Mack and Burdett 1992] is somewhat different from other usability test methods in having an explicit interaction between the test subject and the experimenter (or "coach"). In most other methods, the experimenter tries to interfere as little as possible with the subject's use of the computer, but the coaching method actually involves steering the user in the right direction while using the system.

During a coaching study, the test user is allowed to ask any system-related question of an expert coach who will answer to the best of his or her ability.[9] Usually, the experimenter or a research assistant serves as the coach. One variant of the method involves a separate coach chosen from a population of expert users. Having an independent coach lets the experimenter study how the coach answers the user's questions. This variant can be used to analyze the expert coach's model of the interface. Normally, though, coaching focuses

on the novice user and is aimed at discovering the information needs of such users in order to provide better training and documentation, as well as possibly redesigning the interface to avoid the need for the questions.

The coaching method has proven helpful in getting Japanese users to externalize their problems while using computers [Kato 1986]. Other, more traditional methods are sometimes difficult to use in Japan, where cultural norms make some people reluctant to verbalize disagreement with an interface design.

The coaching situation is more natural than the thinking-aloud situation. It also has an advantage in cases where test users are hard to come by because the intended user population is small, specialized, and highly paid. Coaching provides the test users with tangible benefits in return for participating in the test by giving them instruction on a one-to-one basis by a highly skilled coach.

Finally, the coaching method may be used in cases where one wants to conduct tests with expert users without having any experts available. Coaching can bring novice users up to speed fairly rapidly and can then be followed by more traditional tests of the users' performance once they have reached the desired level of expertise.

6.9 *Usability Laboratories*

Many user tests take place in specially equipped usability laboratories [Nielsen 1994a]. Figure 20 shows a possible floor plan for such a laboratory. I should stress, however, that special laboratories are a convenience but not an absolute necessity for usability testing. It is

9. One variant of the coaching method would be to restrict the answers to certain predetermined information. In an extensive series of experiments, one could then vary the rules for the coach's answers in order to learn what types of answers helped users the most. Unfortunately, this variant requires extremely skilled and careful coaches since they need to compose answers on the fly to unpredictable user questions.

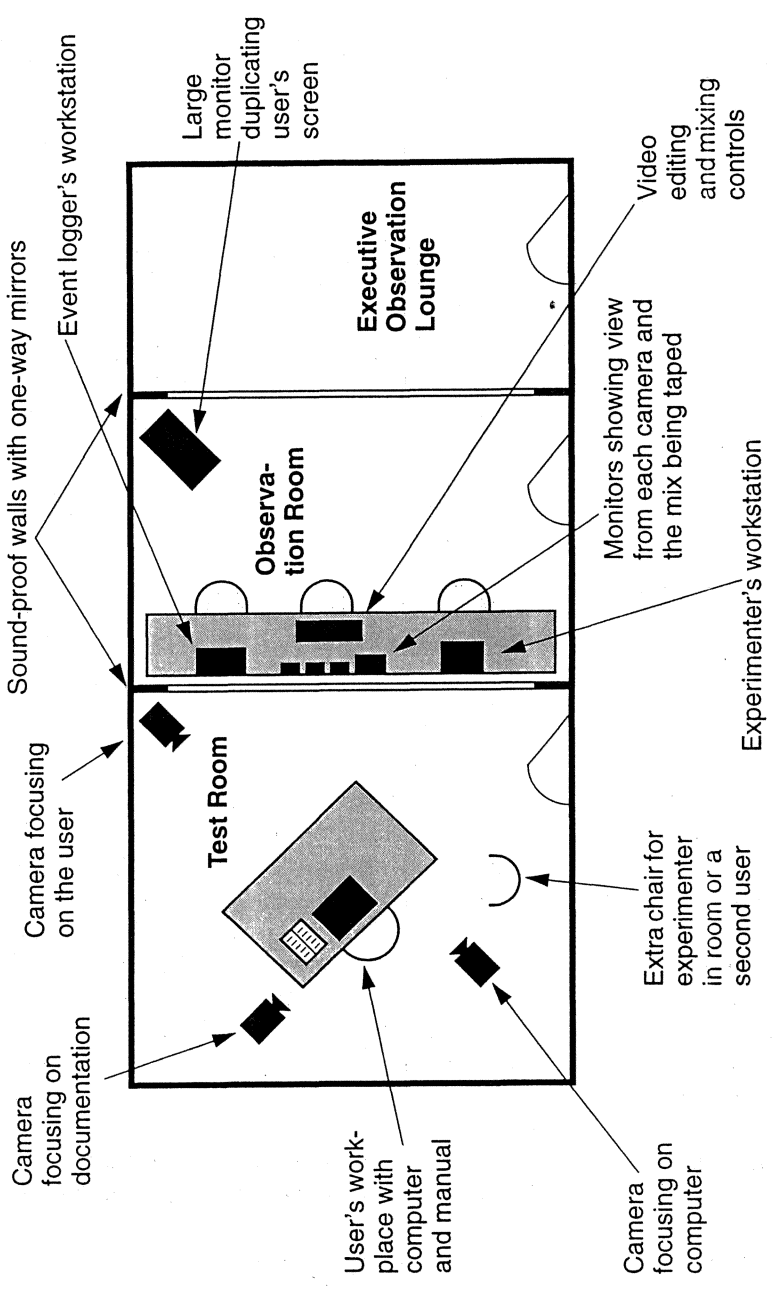

Figure 20 *Floor plan for a hypothetical, but typical, usability laboratory.*

possible to convert a regular office temporarily into a usability laboratory, and it is possible to perform usability testing with no more equipment than a notepad.

In September 1993, I surveyed thirteen usability laboratories from a variety of companies [Nielsen 1994a]. The median floor space of the laboratories was 63 m^2 (678 square feet), and the median size of the test rooms was 13 m^2 (144 square feet). The smallest laboratory was 35 m^2 (377 square feet) with only 9 m^2 (97 square feet) for the test user. The largest laboratory was 237 m^2 and had 7 rooms, allowing a variety of tests to take place simultaneously [Lund 1994]. The largest single test room was 40 m^2 (430 square feet) and was found in a telephone company with a need to test groupware interfaces with many users.

Having a permanent usability laboratory decreases the overhead of usability testing (once it is set up, that is!) and may thus encourage increased usability testing in an organization. Having a special room and special equipment dedicated to usability testing means that there will be fewer scheduling problems associated with each test and also makes it possible to run tests without disturbing other groups.

Usability laboratories typically have sound-proof, one-way mirrors[10] separating the observation room from the test room to allow the experimenters, other usability specialists, and the developers to discuss user actions without disturbing the user. Users are not so stupid that they do not know that there are observers behind a wall with a large mirror in a test room, so one might as well briefly show the users the observation room before the start of the test. Knowing who and what are behind the mirror is much less stressful for the users than having to imagine it. People usually come to ignore unseen observers during the test, even though they know they are there.

Having an executive observation lounge behind the main observation room again allows a third group of observers (e.g., the development team) to discuss the test without disturbing the primary experimenters and the usability specialists in the observation room.

10. One-way mirrors were found in 92% of the labs in my survey.

Typically, a usability laboratory is equipped with several video cameras under remote control from the observation room.[11] These cameras can be used to show an overview of the test situation and to focus in on the user's face, the keyboard, the manual and the documentation, and the screen. A producer in the observation room then typically mixes the signal from these cameras to a single video stream that is recorded, and possibly timestamped for later synchronization with an observation log entered into a computer during the experiment. Such synchronization makes it possible to later find the video segment corresponding to a certain interesting user event without having to review the entire videotape.

More rarely, usability laboratories include other equipment to monitor users and study their detailed behavior. For example, an eyetracker can be used to collect data on what parts of the screens the user looks at [Benel *et al.* 1991].

To Videotape or Not

Having videotapes of a user test is essential for many research purposes where one needs to study the interaction in minute detail [Mackay and Tatar 1989]. For practical usability engineering purposes, however, there is normally no need to review a user test on videotape since one is mostly interested in finding the major "usability catastrophes" anyway. These usability problems tend to be so glaring that they are obvious the first time they are observed and therefore do not require repeated perusal of a record of the test session. This is especially true considering estimates that the time needed to analyze a videotape is between 3 and 10 times the duration of the original user test. In most cases, this extra time is better spent running more test subjects or testing more iterations of the design.

11. The average number of cameras in each test room was 2.2 in my survey, with 2 cameras being the typical number and a few labs using 1 or 3.

Videotape does have several uses in usability engineering, however. For example, a complete record of a series of user tests is a way to perform formal impact analysis of usability problems [Good *et al.* 1986]. Impact analysis involves first finding the usability problems and then going back to the videotapes to investigate exactly how many users had each usability problem and how much they were delayed by each problem. Since these estimates can only be made *after* one knows what usability problems to look for, an impact analysis requires a videotape or other detailed record of the test sessions. Alternatively, one can run more tests and count the known problems as they occur. Impact analyses can then be used to prioritize the fixing of the usability problems in a redesign such that the most effort is spent on those problems that are faced by many users and impact them severely.

Videotape also serves as an essential communications medium in many organizations where it may otherwise be difficult for human factors professionals to persuade developers and managers that a certain usability problem is in fact a problem. Seeing a video of a user struggling with the problem often convinces these people. This goal can also be achieved by simpler means, however, since it is normally even more effective to have the doubter observe a user test in person.[12]

A final argument in favor of videotaping and equipment-extensive usability laboratories is the need to impress upper management and research funding agencies with the unique aspects of usability work. Some usability specialists feel that simpler techniques may not be sufficiently impressive to outsiders, whereas having an

12. Doing so requires strict adherence to the "shut-up" rule: The developers should be advised in advance that they are not supposed to interfere with the user during the experiment. Doing so can be extremely hard for a person who normally has quite strong defensive feelings toward the design. Developers have been known to forcibly interrupt a test user's "maltreatment" of their beloved system and shout, "Why don't you press *that* function key!" This, of course, totally destroys the test. Randy Pausch from the University of Virginia allows developers to be present during user testing but requires them to preface any interruption with the phrase, "I am sorry that I am such a poor programmer that I made the system this difficult to use."

expensive laboratory will result in increased funding and respect due to its "advertising value" [Lindgaard 1991].

Cameraless Videotaping

The main aspects of a test session can be captured on videotape without the use of cameras. Many computers provide a video output that either is directly compatible with video recording or can be made so fairly cheaply by a scan converter.[13] This video signal can be fed directly into the "video in" jack of the video recorder and will thus allow the recording of the exact image the user sees on the monitor. This technique will normally result in better image quality than filming the monitor with a camera, but the video resolution will still be poorer than that of most computer monitors. Furthermore, an audio signal can be fed into the video recorder's "audio in" jack from a microphone, thus creating a composite recording of the screen and the user's comments [Connally and Tullis 1986].

Cameraless videotaping has the obvious disadvantages of not including the user in the picture and not making it possible for a camera operator to zoom in on interesting parts of the screen or the manual page being studied in vain by the user. Unless a high-definition television standard is used, one will also suffer a loss of resolution since current television standards use a poorer quality signal than that used by almost all computer monitors. These limitations may make the resulting videotape less appealing and convincing in some cases. At the same time, cameraless videotaping is considerably cheaper because neither cameras nor operators are needed, and the users are normally less intimidated by a microphone than by a camera.

Portable Usability Laboratories

In addition to permanent usability laboratories, it is possible to use portable usability laboratories for more flexible testing and for field studies. With a portable usability laboratory, any office can be

13. Scan converters were used by 46% of the labs in my survey.

rapidly converted to a test room, and user testing can be conducted where the users are rather than having to bring the users to a fixed location.

A true discount portable usability laboratory need consist of no more than a notepad and possibly a laptop computer to run the software that is being tested. Normally, a portable usability laboratory will include slightly more equipment, however. Typical equipment includes a camcorder (possibly just home video equipment, but preferably of professional quality since the filming of user interfaces requires as high resolution as possible) and a lavaliere microphone (two microphones are preferred so that the experimenter can also get one). The regular directional microphone built into many camcorders is normally not sufficient because of the noise of the computer. Also, a tripod helps steady the image and carry the camera during the hour-long test sessions.

Usability Kiosks

A final approach to the collection of usability data is the *usability kiosk*, which really is a self-served usability laboratory for use as part of a hallway methodology [Gould *et al.* 1987]. In general, the hallway method involves putting a user interface on display in a heavily trafficked area such as outside a company cafeteria in order to collect comments from users and other passersby. A usability kiosk can conduct automated usability testing with self-selected users in such a setting by providing access to a computer running a test interface, suggesting various test tasks to the users, and recording their task times and any comments they might have.

Chapter 7

Usability Assessment Methods beyond Testing

Even though usability testing forms the cornerstone of most recommended usability engineering practice, there are several other usability methods that can and should be used to gather supplementary data. I have already discussed heuristic evaluation in Chapter 5 (page 155) as a method where usability specialists, or even the developers themselves, apply their knowledge of established usability principles to find usability problems without the need to involve users.

7.1 Observation

Simply visiting the users to observe them work is an extremely important usability method with applications both for task analysis and for information about the true field usability of installed systems [Diaper 1989b]. Observation is really the simplest of all usability methods since it involves visiting one or more users and then doing as little as possible in order not to interfere with their work. Of course, the observer can take notes (unobtrusively), and it may even be possible to use videotaping in some environments, though most computer customers do not like to have outsiders come in and videotape their business.