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 question-

naires, 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.

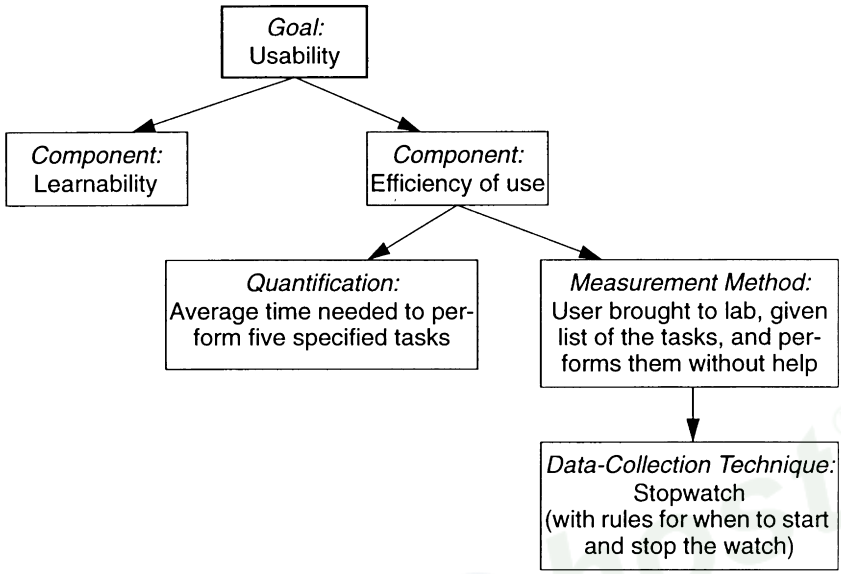

Figure 19 *Model of usability measurement*

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.

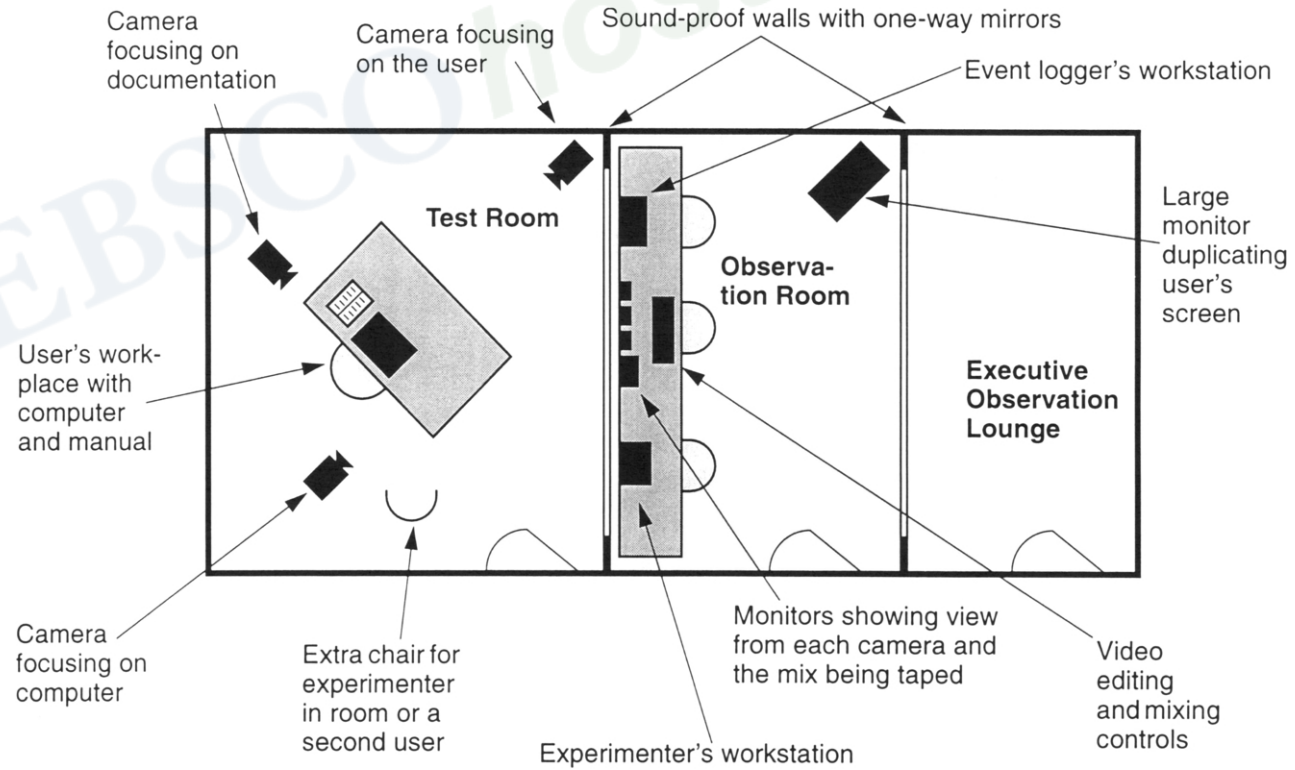

Figure 20 *Floor plan for a hypothetical, but typical, usability laboratory.*