A Practical Guide to Usability Testing

REVISED EDITION

Joseph S. Dumas
Janice C. Redish

intellect

5

Evaluating Usability Throughout Design and Development

Our focus in this book is on usability testing within the context of usability engineering. In the previous chapter, we discussed the importance of that context. In this chapter, we discuss usability testing and other methods to improve usability that you can apply during design and development.

Each of the methods we describe has strengths and weaknesses. The best approach to usability engineering is to develop a strategy that allows you to combine the strengths of several methods. We discuss the strengths and weaknesses of usability testing in this chapter and suggest other methods to make up for its weaknesses. We discuss:

- Getting experts to review the design
- Having peers or experts walk through the design
- Having users work with prototypes
- Getting user edits on early versions of documentation
- Asking users about their satisfaction
- Conducting iterative usability tests

We end the chapter by discussing some recent research comparing usability testing with other usability evaluation methods.

The evaluation methods we discuss in this chapter are not the only methods you can use, but they are commonly used and they have been evaluated in a series of research studies.

Getting Experts to Review the Design

One of the traditional methods that American companies use to get help about a special area of knowledge is to "call in an expert." Evaluating usability is one of the areas where this method is used. It is not unusual for a company developing a product to seek consultants who profess to be experts in usability. Are these experts effective? The answer to that question is not simple because it provokes other questions, such as:

- Does it work compared to what?
- Does an expert do a better evaluation of usability than the product's designers?
- Is using an expert cost effective?
- On what basis does an expert evaluate usability?
- What are the qualifications of an expert?

Recently, some research has addressed some of these questions. The primary motivation behind these studies has been to find usability evaluation methods that are cost-effective. One of the assumptions that is often mentioned in the introductions to these studies is that usability testing is an expensive method and, therefore, organizations

that develop products are looking for cheaper ways to evaluate usability. For example, Desurvire, Kondziela, and Atwood (1992) open their paper on comparing usability evaluation methods by saying, "There is increasing interest in finding usability testing methods that are easier and cheaper to implement than traditional laboratory usability testing, which is frequently not performed due to the lack of funds, planning, or human factors expertise."

As we will see later in this chapter, the question of whether usability testing is expensive compared to other methods depends on a number of factors. In this section, we will focus on the effectiveness of using "experts" to evaluate usability and whether software engineers can be trained to evaluate the usability of products.

The term *heuristic evaluation* describes a method in which a small set of evaluators examine a user interface and look for problems that violate some of the general principles of good user interface design we described in Chapter 4. In one of the early heuristic evaluations, Nielsen and Molich (1990) looked at the effectiveness of training computer science students to evaluate usability. Nielsen and Molich described nine basic usability principles to the students and then had the students find violations of the principles in a user interface. The intention here was to provide these students with a set of heuristic rules, nine basic usability principles, to evaluate the usability of the product. The nine principles were:

1. Use simple and natural language.
2. Speak the user's language.
3. Minimize user memory load.
4. Be consistent.
5. Provide feedback.
6. Provide clearly maked exits.
7. Provide shortcuts.
8. Provide good error messages.
9. Prevent errors.

The motivation for this investigation was to look for a method that might be cheap to use. Giving software engineers a short tutorial on usability principles is cheaper than having designers teach usability specialists about their product and then have the specialists review the product and tell the designers where the problems are.

This first study showed that, as individuals, the student evaluators did not find many of the usability problems with the products they evaluated. The range of problems each student found was between about 20% and 50%. Nielsen and Molich then statistically aggregated the individual evaluators into groups of varing sizes and found that groups of five evaluators working separately would have found between about 50% and 75% of the usability problems. On the basis

of this data, Neilsen and Molich recommended using heuristic evaluation with three to five evaluators.

Since that 1990 study, several additional studies have looked at the effectiveness of heuristic evaluation.

Jeffries et al. (1991) took a different approach to what they called heuristic evaluation. They used people with advanced degrees and experience in usability engineering as their experts. They asked these people to find the usability problems with the interface to a product without specifying any set of general principles to follow. These authors also had a group of software engineers read a report describing a set of 62 usability guidelines and evaluate the usability of the same product.

The usability experts found about three times as many problems as the software engineers and more than twice as many of the most severe problems. Since Jeffries et al. did not have an absolute measure of the number of usability problems with the product being evaluated, we cannot look at the success rate of these evaluators. Jeffries et al. recommend using software engineers to evaluate the usability of a product only when there are no experts in usability available.

It is difficult to compare the Jeffries et al. study with Nielsen and Molich because of the different definitions of heuristic evaluation. The software engineers whom Jeffries et al. had read the 62 usability guidelines are more comparable to the software engineers Nielsen and Molich used. In both studies, their success at finding usability problems was disappointing. In a related study, Desurvire et al. (1992) found that software engineers who were given a lecture on usability principles found less than half the problems that usability specialists found, but did find many specific, local problems.

The four usability experts that Jeffries had evaluate the usability of the interface have some unknown, unmeasured level of expertise in human–computer interaction. The lack of information on their expertise makes it difficult to compare their performance with other groups of "experts." The studies we describe shortly all have this weakness.

Nielsen (1992) used three groups of evaluators:

1. "Novice" evaluators, who had no training or experience in usability engineering.
2. "Regular" usability specialists, who were experts in usability engineering but with no special expertise in the usability of the type of interface they were evaluating—a telephone voice-response system.
3. "Double" usability specialists, who were experts in usability engineering and who also had experience working with telephone voice-response systems.

The novices found only 22% of the usability problems, the regular usability specialists found 41%, and the double usability specialists found 60%. Nielsen also statistically aggregated the individual evaluators into groups and found that five novice evaluators would

find 50% of the usability problems, five regular usability specialists would find about 80%, and five double usability specialists would find about 98% of the problems.

These studies taken together show that software engineers are not very effective at finding usability problems. The viability of giving software designers a quick lecture or having them read a report on usability principles and be effective at finding usability problems is questionable. Some researchers are still optimistic about the potential of this concept, but the evidence produced to date is not convincing.

As you might expect, experts in usability engineering are much better at finding usability problems than software engineers and usability experts who have experience with the technology they are evaluating are best. The studies that report this finding, however, are difficult to compare, because they have no measure of what it means to be an expert.

While it is difficult to compare these studies, there is a finding that is consistent: Heuristic evaluation, when used by experts, tends to uncover many local problems. In the Jeffries et al. study, the experts using this method found 52 problems that were judged "least severe." Usability testing uncovered only two of these problems.

In the final section of this chapter, we will talk more about this finding, but it is worth noting here that conducting both a heuristic evaluation with experts and a usability test would allow you to combine the strenghts of these two methods: Usability testing would uncover most of the major, global problems, and heuristic evaluation would uncover most of the minor, local problems.

Research issue:	It would be easier to compare studies using experts if we knew more about how the experts were uncovering problems. Nielsen's study (1990) suggests that just as "double" usability specialists have some knowledge they have gained from working with the technology underlying the interface, "regular" usability specialists have some knowledge they have gained about human–computer interaction. It would help us understand usability evaluation much better if we had a large number of "usability experts" talk aloud as they all evaluated the same interface. We could compare their approaches and see if there is any consistency among them.

In the final section of this chapter, "Comparing Usability Testing with Other Usability Evaluation Methods," we will look at how heuristic evaluation compares with usability testing as a cost-effective way to evaluate the usability of an interface.

Having Peers or Experts Walk Through the Design

Structured walkthroughs have become a standard method for assuring the quality of software (Yourdon, 1985). Walkthroughs are a peer-group review of a technical product. They can be used to review specifications, designs, or programming code. During a walkthrough, the team of people who are developing a product "walk through" the specifications or the programming code one step at a time looking for bugs or inconsistencies.

There is a variation of the walkthrough, called a cognitive walkthrough, designed to evaluate the usability of the user interface (Lewis & Polson, 1990). Its creators have proposed that this method is an effective way to evaluate a user interface when a full prototype is not feasible. The goal of the method is to evaluate the strengths and weaknesses of the user interface without having to create a prototype or the design itself.

The cognitive walkthrough is based on a formal model of human–computer interaction. The model assumes that people learn about an interface through a complex guessing strategy. Users make guesses about what actions to take by comparing the expected outcome of the action against their goals. They then take action and evaluate their progress toward the goals.

The walkthrough consists of answering a set of questions about each of the decisions users must make as they use an interface. The questions have to do with identifying users' goals, the ease with which users will be able to identify the consequences of a decision, and how easy it is for users to evaluate whether they are making progress toward a goal. These questions are asked for each step of each task.

After those taking part in the walkthrough answer the questions for each step, they rate the likelihood that users will have problems making the correct choice or action. Those steps at which users will be most likely to have problems need improving.

One of the advantages of this method is that it makes users' goals and expectations explicit. When a step is likely to cause problems for users, the walkthrough indicates where the problem lies, for example, in the feedback the system gives after users have taken some action.

While the walkthrough can yield useful diagnostic data, it has been tedious to perform. The designers we have worked with would not have the patience to make this method work in its paper-and-pencil form. There is a software version of the method, called the Automated Cognitive Walkthrough, that takes much of the tedium out of the process of using the walkthrough (Desurvire et al., 1992). In addition, researchers continue to simplify the cognitive walkthrough to reduce the time it takes to perform as well as the tedium of the process (Rowley & Rhodes, 1992).

Lewis and Polson (1990) propose that this method can be used by designers who do not have expertise in human–computer interaction, but we are skeptical. The validity of the method depends on an accurate and thorough understanding of who the users are and what skills and experience they have. As long as the designers have this understanding, the walkthrough may be useful. But, if the designers have an incorrect view of the users of the product, the walkthrough will be invalid.

There have been some recent research studies looking at the effectiveness of walkthroughs (Desurvire et al., 1992; Jeffries et al., 1991). Generally, walkthrough methods are less effective at finding usability problems than other evaluation methods, such as heuristic evaluations. Karat, Campbell, and Fiegel (1992) have shown that the effectiveness of walkthroughs can be enhanced by conducting the walkthrough in a group rather than with individual evaluators.

In the final section of this chapter, "Comparing Usability Testing with Other Usability Evaluation Methods," we will look at how walkthroughs compare to usability testing as a cost-effective way to evaluate the usability of an interface.

Having Users Work With Prototypes

Prototyping is not a new idea. It is a valuable tool that was first used in developing hardware. In that context, a prototype was a handmade, mechanical model of a design. It gave both designers and users a hands-on feel for the product. These hardware prototypes varied in their fidelity to the final product, from simple foamcore mock-ups to near-production models.

Until recently, designers of software user interfaces did not often use prototypes. There was no quick and easy way to simulate the look and feel of software. Creating a prototype of any complexity was just as difficult as creating the code for the final product. Consequently, designers could not conceive of creating a user interface that was separate from the rest of the software (Edmonds, 1992).

Let's look at two types of prototyes:

- Static, paper-based prototypes
- Interactive, software-based prototypes

Then we will compare the effectiveness of these two types of prototypes and look at some of the strengths and weaknesses of prototypes.

Using Static, Paper-based Prototypes

One of the ways to get users involved in design early is to show them screen images on paper of what a product will look like and ask them to try them out. These paper images are sometimes called a

screenplay. A screenplay is a static prototype that can give you valuable information about the usability of your product.

A screenplay can be effective particularly at helping you solve problems such as developing a menu hierarchy that users can understand. Let's look at an example to see what we mean.

This is the top level menu for a hypothetical drawing program:

File	Edit	Draw	Font	Options	Window	Help

While this example of a drawing program is hypothetical, we have constructed it from some actual drawing programs we have seen.

Imagine that you have drawn a figure. You look at it and decide that you want to make the lines in your figure thicker or darker. Which menu option would you go to? Most people we have asked choose "Draw." Here is the submenu for Draw:

Draw
Palettes...
Front
Back
Group
Degroup
Grid on
Grid off
Snap to grid

People are generally bewildered by some of these choices, but decide to go back to the top level and choose another option. They often choose Edit next, but quickly decide the right choice isn't there:

Edit
Can't undo
Repeat
Cut
Copy
Paste
Select all
Select block

Next, they'll often go to Options:

Options
Preferences...
Tools...
Zoom
Unzoom
Transform...
Import
Export

From here, people usually choose Tools, but that brings up a palette for doing new work. They usually then leave this menu, flounder around in other choices for a while, go back to Draw and Edit, and after several more wrong moves, decide they can live without changing the lines. The correct choice would be Transform on the options menu. That brings up a dialogue box that permits many different changes to a drawing.

You can see how easy it is to find usability problems with menu hierarchies by making a screenplay and having users work through it. This happens to be the kind of usability problem that paper prototypes are good at revealing (Nielsen, 1990). But this was only one problem, and there may be many more. Using the static screens is slower than using an interactive software prototype, but it can work. (See the later discussion, however, in the section "Comparing Static and Interactive Prototypes" for some further limitations of paper prototypes.)

As the size of the menu hierarchy grows, a screenplay becomes more cumbersome to use, but it is useful for problems such as the effectiveness of a small menu hierarchy.

Interactive, Software-based Prototypes

Over a very short period, about five years, most of the impediments to creating useful software prototypes have disappeared. It is now possible to simulate the look and feel of a software user interface in a few hours, or a few days for a more complex product. Furthermore, prototyping tools are now available for most hardware and operating system environments.

Usability tests of prototypes of a design are becoming very common, because they allow designers to make changes before it is too late. Most software applications are sufficiently complex that it is

difficult for designers to understand how the product will work unless they create a prototype.

Ironically, software prototyping tools are also being used to display images of hardware user interfaces. For example, we have created software prototypes of a remote control for a TV and a control panel for a complex hospital bed. While these prototypes do not help users to touch or feel the product, they can present realistic views of layouts, buttons, and labels. These prototypes allow designers to get feedback on usability from users long before they create actual hardware prototypes.

The Benefits of Interactive Prototypes

In a very real sense, user-driven software design and usability testing of software would be very difficult without prototyping tools. There are four major benefits of software prototypes:

1. They make it possible to incorporate user feedback into the design early in the development process.
2. They allow designers to explore several design concepts before they settle on one.
3. They make it possible to evaluate several iterations of a design.
4. They take the fuzziness out of what the user interface is and, therefore, allow members of the design team to communicate with each other about the user interface.

This fourth benefit is worth elaborating. With an interactive prototype, a software engineer, who is involved in the details of a design, and a marketing manager, who is only peripherally involved, can use a prototype of the user interface to talk about it. The prototype reduces the likelihood that people will miscommunicate about how the user interface will look and operate.

Prototyping Less Than the Full User Interface

Prototypes are often created to explore part of the interface. For example, a design team might create three prototypes that represent different design concepts. Each prototype might mimic only a small part of the user interface. Nielsen (1989, see Figure 5-1) makes a distinction between types of prototypes that each represent only a part of the user interface:

- Horizontal prototypes reduce the size of the prototype by containing a shallow layer of the surface of the user interface. For example, the prototype might show a "desktop" screen with the names of each main menu item or

icons on it. For an electronic mail horizontal prototype, users might see an opening screen with a list of messages and the main menu showing the options available. If users selected a menu option, they would receive a message indicating that further selections are not implemented or perhaps just hear an audible tone.

- Vertical prototypes fully implement a small number of paths through the interface, but do not include any part of the remaining paths. For example, all of the branches of a "Create" option of an electronic mail product might be implemented so that the user could edit, transmit, store, or print a message.
- Scenario prototypes differ from the other two types by being task oriented. A design team might decide to fully implement three important tasks that cut through the functionality of the prototype. For example, the user might be able to create a message, send it, and store it in a folder. If the user deviates from the preferred path or selects an incorrect option, that part of the prototype would not be implemented.

Nielson recommends the use of scenario prototypes as a tool for evaluating usability. The limited size of these prototypes makes it easy to change them frequently on the basis of some form of feedback from users.

There is a potential problem with partial prototypes. Because they mimic only part of the user interface, partial prototypes may lead a test team to overestimate the usability of a product. In a partial

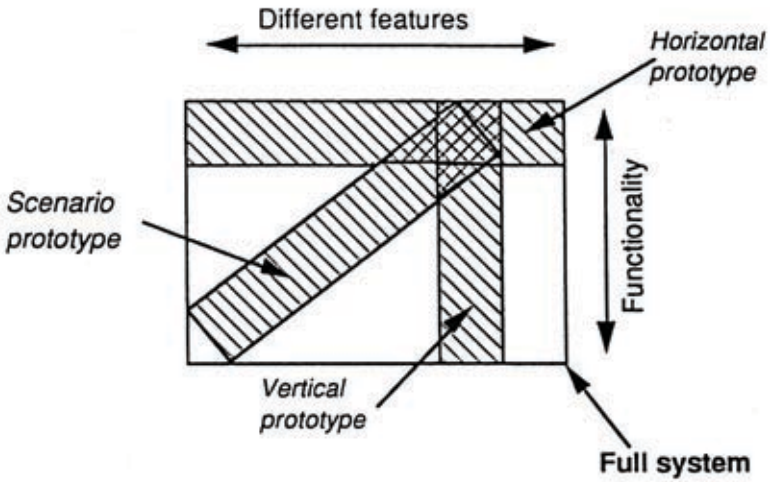

Figure 5-1. Horizontal,vertical,and scenario prototypes (modified from Nielsen, 1989)

prototype, there are limited ways in which test participants can go astray. When a participant selects an option in a software prototype that has not been implemented, the prototype usually displays a message informing the participant that the option is not part of the prototype. In the real product, users will be able to select that option and could, by doing so, get into more trouble than they can when the option is not available. When a user interface has several global problems that cut across the user interface, the fully implemented user interface presents more ways for test participants to go wrong.

Comparing Paper Versus Interactive Prototypes

Nielsen (1990) has compared the effectiveness of using interactive prototypes with the static paper prototypes we described earlier in this chapter. He had two groups of evaluators examine either a paper version of a prototype of a user interface or an interactive software-based version created with a prototyping language.

There were 50 usability problems with each interface, 15 of which Nielsen called "major." These major problems are what we call "global." The results show that the evaluators who were using the software prototype found significantly more global problems. The one global problem that the paper prototype was effective at uncovering was a lack of consistent navigation rules in a menu hierarchy.

The results of this study suggest that when you have access to interactive prototyping tools, it is preferable to use them in place of paper prototypes.

Some Cautions About Prototypes

There is a lively debate in the human factors community about how close the prototype needs to be to the final product to be a useful tool for usability testing. The value of the new prototyping tools is that they allow designers to try out design concepts quickly (Melkus & Torres, 1988; Virzi, 1989). But they also allow designers to use the prototype to create the entire user interface and to make it look like a finished product. Used in this way, the prototype becomes a product of its own, a demanding product that requires complex debugging.

We have seen design teams who want the prototype to mimic the complete user interface and to look like a finished product. They are afraid to show users a product that looks like it is still under development because it may bias users against their design. In our view, these designers waste valuable time treating the prototype as if it were a product itself. Prototyping tools are best used to explore alternative concepts. They do not have to have high fidelity to be useful.

Another limitation of software prototyping tools is the difficulty of simulating the response time that the real product will have. With a prototyping tool, you can create a realistic looking and acting user

interface, but often you cannot predict how long it will take for the final product to perform such actions as looking up information in a database or drawing a complex graphic. The prototype, which does not actually carry out these actions, may respond immediately to a command while the final product may respond more slowly. Consequently, the reactions you get from participants during a usability test of a prototype can overestimate the ease of use of the final product. If you can anticipate the length of any delays, build them into the prototype. (See Hix & Ryan, 1992, for a quantitative method for evaluating the effectiveness of prototypoing tools.)

Research issue:	Do prototypes that are low in fidelity bias users to rate them lower in usability than a prototype of higher fidelity? How much does the appearance of the prototype contribute to users' ratings of its quality or ease of use?

Getting User Edits on Early Versions of Documentation

Just as you can have users try out prototypes of the user interface, you can have them try out drafts of the documentation. The documentation might be an installation card, a quick reference card, the outline and/or sections of the users' manual, or samples of the online help.

Testing samples of draft documentation serves exactly the same purpose as testing prototypes of the interface. It allows the writers to see if they are on the right track with the content, organization, language, and layout of the documentation from a small piece of the work. Changes at an early stage save time and money, just as they do in developing the interface.

Depending on the status of the documentation and the rest of the product, getting users involved at the early stages can range from asking a few users to read and comment on parts of the documentation to including the draft documentation with the prototype of the interface in a usability test. As with any other aspect of usability engineering, a critical factor in getting users involved is making sure that the people whom you have work with the draft documentation represent real users.

Here are a few ways to involve users with draft documentation early on:

If you are concerned about whether the organization of the draft document matches the way that users will approach it, you can do a test

very similar to the static prototype of the menu structure of an interface. You can give users tasks and ask them what they would look up in the manual or online help when trying to do that task. You can give them a draft table of contents to use. A draft table of contents is the outline of the manual, reflecting all the headings and subheadings that are planned for the manual.

If you are concerned about whether the content is what users need and whether the style (language) will be clear to users, you can prepare the documentation for one part of the product and see how well that works for users. With the desktop publishing resources available today, preparing draft printed documentation that looks very much like the final documentation is easy.

Getting online help into a prototype may be more difficult. If you are prototyping in the same system that will be used for the final product, Microsoft Windows®, for example, you can probably have the online help work with a prototype. If you are prototyping with a different system, and the way help works in that system is not the same way it will work in the final product, you would probably not want to put the online help into the prototype. You could still have samples of the online help available on paper.

Atlas (1981) described a *user edit* as having a user do a task, using the instructions for the product as a guide. An observer watches, listens, and takes notes. If you have the user try to follow the instructions, you can use this method to find out if that particular set of instructions is accurate, complete, and clear to the user. If the user comments on problems, or if the user hesitates, misreads, or makes mistakes, you have indications of problems in the text. To the extent that the instructions the user is trying out are similar in level of content and style to the rest of the manual, you have information that can lead to much broader changes.

As Redish and Schell (1989) point out, this type of user edit can give you very useful information about the general style and level of detail of the instructions, but it may not help you understand how well the document as a whole is organized for users. If the user edit is of a specific part of a document and the user goes through the document page by page and step by step, you are not seeing how easy or difficult it is for users to *find* the information that they need in the manual. You are, however, seeing how useful the manual is once the users have found the correct page.

Soderston (1985, p. 18) described a *usability edit* as giving a user "the task for the day, our written material, and the system." Observers watch and perhaps videotape the session. To the extent that "our written material" includes enough of the manual to make users both search for the correct information and then read (use) it, Soderston's usability edit can yield information about the organization as well as the content and language of the manual.

A major difference between Atlas's user edit and Soderston's usability edit is that, in the latter, the user is told to think out loud throughout the session. In essence, Soderston is describing a usability test of part of the product and the documentation for that part.

Having users think out loud while performing any task, from reading a text to working with a product, is also called having the user *give a verbal or think-aloud protocol.* Since the early 1980s, think-aloud protocols have been used very successfully in understanding users' problems with a wide variety of documents (Schriver, 1989, 1991). As Schriver (1991, p. 167) points out, "very often, protocols will help writers detect both problems of commission, that is, problems caused by what the text says, and problems of omission, that is, problems caused by what the text is missing."

When testing draft documentation as part of any usability test, however, keep these two caveats in mind: The first caveat is that it is very difficult to test both *whether people will* use the documentation and *how* they will use it in the same test. The second caveat is that you must make the measures that you take about the documentation match the stage of development that the documentation is at. For example, if the draft manual does not yet have either a table of contents nor an index, taking a measure of "time to find the information users need" doesn't make much sense.

Conducting Iterative Usability Tests

In Chapter 2, we discussed conducting usability tests on all parts of a product—the software and hardware user interface and the documentation—and conducting tests early and often. We are advocating an approach to product design in which designers expose a product under development to users in the form of usability tests as early as possible and continue to conduct tests a often as necessary to ensure the usability of the product.

We also advocate combining usability tests with the other evaluation methods we discussed in this chapter. In the next section, we describe which evaluation methods work best with usability testing.

Comparing Usability Testing with Other Usability Evaluation Methods

In the past two years, there have been a number of research studies comparing the effectiveness of usability testing with some of the other usability evaluation methods we have discussed in this chapter. We expect that there will soon be many more studies, and they will

provide additional information about the advantages and disadvantages of each method.

We briefly discuss the recent studies here. In general, they show that usability testing compares well with other evaluation methods. There are enough differences in methodology among these studies, however, that we need additional research to give us a better understanding of the relationships between the evaluation methods.

The Jefferies et al. Study

Jeffries et al. (1991) compared the relative efffectiveness of four evaluation methods at uncovering usability problems in a software application:

1. Applying guidelines—Software engineers were given a report describing 62 guidelines of good practice in usability. The engineers studied the guidelines and then used them to evaluate the software user interface.
2. Heuristic evaluation—Four evaluators who had training and experience in human–computer interaction evaluated the software user interface. These experts were not told what basis they were to use to conduct the evaluation.
3. Cognitive walkthroughs—Software engineers working as a group were taught how to do a walkthrough of the user interface. They then conducted a group walkthrough of the product and identified usability problems.
4. Usability test—A human factors specialist conducted a usability test with six participants and identified usability problems.

There were several interesting results from this study. The heuristic evaluation found the most problems (105), compared with applying guidelines (35), the walkthrough (35), and the usability test (31). However, no single expert found more than 42 usability problems in the heuristic evaluation.

Jeffries et al. had seven usability specialists rate the severity of the problems that these methods found. Jeffries et al. then ranked the problems on the basis of the severity rating. Figure 5-2 shows the top and bottom thirds of problems ranked on severity.

As you can see, the heuristic evaluation found the most problems, although its advantage over the usability test is much smaller with the most severe problems. The experts listed a large number (52) of least severe problems. Jeffries at al. note that they are not sure that all of these problems really need to be fixed. Notice, also, how few of the least severe problems were uncovered by the usability test. This is not surprising, because tasks selected for a test are intended to sample the

Level of Severity of Problems	Guidelines	Heuristic Evaluation	Cognitive Walkthru	Usability Test
most severe	12	28	9	18
least severe	11	52	10	2

Figure 5-2. Level of severity of problems found by each evaluation method. (From Jeffries et al., 1991)

parts of the interface that are most likely to create global problems. If a local problem occurs on a screen that test participants never use, the test will not uncover the problem.

The results of this study show clearly the strengths and weaknesses of these methods.

Usability testing finds global problems very well but is poor at uncovering local problems. Heuristic evaluation, on the other hand, finds many specific, local problems. It would appear, therefore, that heuristic evaluation and usability testing nicely complement each other.

Usability testing finds global problems very well but is poor at uncovering local problems.

Jeffries et al. computed a cost-benefit analysis of the four methods. Again, heuristic evaluation yields the highest payoff. The four experts took a total of 20 hours to do their evaluation; the usability test took nearly 200 hours.

This study shows the value of having more than one expert review the usability of a design. Clearly, if you have four experienced usability specialists available to you, asking them to independently evaluate the interface is the most cost-effective evaluation method. If you do not have usability experts available, the only other method that uncovers a substantial number of severe usability problems is usability testing.

An interesting question that arises from this study is whether the usability test uncovers problems that the other methods do not find. Jeffries et al. did not conduct this analysis, but they report, as an anecdote, that a severe problem that resulted in users being unable to log into the system after deleting a directory was uncovered by one of the participants in the usability test. This problem was not uncovered by any of the other methods.

The Bailey et al. Study

While using both heuristic evaluation and usability testing may prove complementary, Bailey, Allan, and Raiello (1992) have shown that changing many of the local problems that are uncovered with heuristic evaluation did not improve the usability of a software user

interface. They had experts list the usability problems with a user interface. The experts uncovered 43 usability problems with the product. Bailey et al. also conducted a usability test with the product. They fixed the single most serious problem with the product and retested it. They then fixed the single most serious usability problem with the product and retested it. They continued with this procedure of fixing a usability problem and then retesting two more times. The results showed that there was no statistically significant improvement in preformance after the two most serious problems were fixed.

This study suggests that many of the specific problems that are uncovered in a heuristic evaluation do not influence the performance of users. We do not know whether these small problems would influence users' perception of the ease of use of a product.

The Karat et al. Study

Karat et al. (1992) conducted a study comparing usability testing to walkthroughs. In this study, two different products were evaluated to assess the reliability of the methods. In addition to having six individual evaluators conduct a walkthrough, they also had six pairs of evaluators conduct a walkthrough together to see if walkthroughs are made more effective when there is group interaction. The usability tests had six test participants using the products.

The results show that the usability tests uncovered about twice as many problems as the group walkthroughs and three times as many as the individual walkthroughs. The usability tests also uncovered significantly more severe problems than the walkthroughs.

Figure 5-3 shows the number of *unique* usability problems found by each method, that is, finding a problem that none of the other methods found. As you can see, the usability test uncovered many more unique problems than the other methods. About two-thirds of these problems were severe.

Karat et al. also conducted a cost-benefit analysis of the methods. As you might expect, the usability tests required the most time to conduct, but required less time-per-usability problem than the other methods.

	Usability Tests	Team Walkthru	Individual Walkthru
System 1	13	1	0
System 2	8	0	2

Figure 5-3. Unique usability problems uncovered by each method. (From Karat et al. 1992)

The Desurvire et al. study

Desurvire et al. (1992) compared usability testing with heuristic evaluation and cognitive walkthroughs. In addition, they had three different types of evaluators: usability experts, software engineers and nonexperts.

One of the unique features of this study was that the software engineers had designed the product they were evaluating. One might think that software developers who understand the internal workings of a product will be able to find more usability problems than software designers who are evaluating a product they did not design.

Consistent with Karat et al., the usability test in the Desurvire et al. uncovered the most problems, and the usability experts found more problems than the nonexperts or the software engineers. Figure 5-4 shows the number of problems that were uncovered by each of the conditions.

As you can see, the usability test found more than twice as many problems as the heuristic evaluation, which, in turn, found more problems than the cognitive walkthrough. The usability experts found about twice as many problems as the software engineers, who, in turn, found about twice as many problems as the nonexperts.

Comparing the Studies with Each Other

The one major inconsistency among these studies is the effectiveness of the performance of experts using heuristic evaluation. Jeffries et al. found that the experts uncovered more problems than the usability test did, while other studies found that the experts uncovered less than half of the problems uncovered by a usability test. It is difficult to compare these studies on these factors, because they each used

	# of Problems	% of Problems
Usability test	25	100%
Heuristic Evaluation		
Experts	11	44%
Software Engineers	4	16%
Non-Experts	2	8%
Cognitive Walkthrough		
Experts	7	28%
Software Engineers	4	16%
Non-Experts	2	8%

Figure 5-4. Number and percent of usability problems uncovered by each method (From Desurvire et al. 1992)

different experts and did not measure the expertise or experience of the experts in any systematic way.

One of the consistent findings of these studies, however, is that heuristic evaluation when conducted by experts uncovers many more local problems that other methods. Usability testing does just the opposite, that is, it uncovers the global problems.

Taken together, these studies suggest these conclusions:

- Usability testing uncovers more usability problems than other evaluation methods.
- Usability testing finds more global problems than other evaluation methods.
- Usability testing finds more unique problems than other methods.
- Usability testing uncovers fewer local problems than other evaluation methods.
- Usability testing takes more hours to conduct than other methods, but is cost effective when considered on a cost-per-problem-uncovered basis.
- Heuristic evaluation, conducted by usability specialists, is better at uncovering usability problems than walkthroughs.
- Heuristic evaluation gains in power when there are several usability experts working independently.
- Heuristic evaluation uncovers more minor problems than other methods, but changing these minor problems may not improve performance.
- Cognitive walkthroughs are less effective than heuristic evaluation and usability testing at uncovering usability problems.
- Software engineers are not very good at uncovering usability problems, even when they are given a short lecture or report on principles of human–computer interaction.

Our recommendation, at this time, from the available evidence is to conduct both usability tests and heuristic evaluations to take advantage of each method's strengths. As Jeffries and Desurvire (1992) note, "the best evaluation of a user interface comes from applying multiple evaluation techniques." Usability tests will uncover the global problems and will also uncover more problems that the other methods miss. If you only conduct usability tests, however, you run the risk of having many undetected local problems in your product. While any one of these problems is probably not serious enough to keep users from completing tasks or frustrating them, the combined effect of having many local problems is likely to make the user feel that the designers were sloppy in their development effort.

In the next chapter, we discuss how to integrate usability testing into an organization that develops computer-based products.