Heuristic Evaluation of User Interfaces versus Usability Testing

Soren Lauesen and Mimi Pave Musgrove

This is Chapter 14 from Soren Lauesen: User Interface Design - A Software Engineering Perspective, Addison-Wesley 2005, reprint 2007.

The chapter contains original research, but was never published in a journal or a conference proceeding. To make it widely available and promote the textbook, Soren Lauesen has published it on his web-site.

Section 14.4 is particularly interesting. It compares 17 evaluations of the same user interface, a hotel booking system for Hotel Pennsylvania, New York. The evaluations were made by 17 top-level usability teams in an experiment conducted by Rolf Molich and Robin Jeffries. Eight teams used heuristic evaluation and nine teams usability tests. The teams performed very differently, for instance in the number of problems they identified. Molich and Jeffries made various subjective comparisons, but never concluded anything about the basic question.

In contrast, Lauesen and Musgrove made a careful statistical comparison. It turned out that there were no significant differences between the two groups. The differences have other causes than the technique used.

The comparison uses statistics that is intuitive and rather easy to understand. It shows the numbers behind the calculations, how randomness works (the statistical model) and how the calculations were made in Excel.

14

Heuristic evaluation

Highlights

- Various kinds of heuristic evaluation.
- Cheaper than usability testing? Not necessarily.
- Missed problems and false positives? It depends.
- Users are different and randomness is inevitable.
- A precise statistical comparison of 8 expert teams against 9 usability teams (CUE-4).

Heuristic evaluation and usability testing are two different techniques for finding usability problems. With heuristic evaluation, someone looks at the user interface and identifies the problems. With usability testing, potential users try out the user interface with real tasks. The problems found with usability testing are *true* problems in the sense that at least one user encountered each problem. The problems found with heuristic evaluation are *potential problems* – the evaluator suspects that something may be a problem to users. Early in development, heuristic evaluation has a hit-rate of around 50% and reports around 50% false problems. This is the first 'law' of usability (section 1.5).

In the world of programming there are similar techniques. During a *program review*, someone looks at the program text to identify bugs. This corresponds to heuristic evaluation. During a *program test*, the program is run on the computer and the programmer checks the results. This corresponds to usability testing. Good program reviewers can identify around 90% of the bugs and they report few false bugs.

14.1 Variants of heuristic evaluation

Heuristic evaluation may vary according to the way the system is introduced to the evaluators, the way the evaluators look at the system, the evaluator's background, etc.

Introducing the system

Explain the screens. The designer may explain each of the screens to the evaluator before the evaluation. This of course helps the evaluator so that he may miss some of the problems that real users might encounter. If the designer explains a lot, the exercise may turn into a design review where designer and evaluator discuss problems and alternative designs.

Explain nothing except the purpose of the system. If the designer doesn't explain the screens but only the purpose of the system, the evaluator will be on his own in a way similar to a user. This approach is suited for systems to be used without introduction or training. However, if the system is reasonably complex, the evaluators need some introduction to the system. As a result the evaluator may miss some problems. Experienced developers try to strike a balance by giving only a brief introduction to each screen.

Method

Heuristic rules. The evaluators may use a list of rules (guidelines) for identifying potential problems. They look at the screens one by one, trying to determine where the rules are violated. No doubt, rules help identify potential problems that otherwise would have been overlooked. However, if the list of rules is long, evaluators are unable to check all the rules carefully. We will look at some examples in section 14.2.

Subjective judgement. The evaluators may look at the screens one by one using their subjective judgement and earlier experience. This is the typical approach, particularly if the evaluators are potential users. Even if the evaluators use heuristic rules, they cannot suppress their subjective judgement, so it is hard to isolate the effect of the heuristic rules.

Task-based evaluation. The evaluators may be asked to check how various tasks are carried out – or they may define such tasks on their own initiative. With this approach, the exercise becomes quite similar to a usability test. The main difference is that the test subject (the evaluator) also records the problems. This variant may have a hit-rate close to usability testing as we will see in section 14.4.

Unfortunately, the task-based evaluation assumes that the system is operational. If the system is still a mock-up, the evaluator doesn't know how the system will react (the human 'computer' is not there). As a result, the evaluator may believe it works in one way and he will not realize that it is intended to do something else.

Evaluators

Usability specialist. The evaluator may be a usability specialist – a person with knowledge of usability principles and experience in using them. However, the specialist has no expertise in the application domain. Usability specialists tend to report more potential problems than other kinds of evaluators. However, they still miss around 50% of the true problems when using heuristic rules or subjective evaluation.

Fellow developers. Developers without specific usability expertise can serve as evaluators, but most of the results I have seen are not good. Fellow developers tend to focus on technical details and deviations from the user interfaces they develop themselves. They tend to suggest changes rather than pointing out the usability problems. If they discuss their findings with the designer, they easily end up in technical debates about what is possible and what should be done. (I often hear about usability courses for developers where participants review each other's designs. Usually it ends in a disaster, particularly if they all try to design the same system.)

Potential users. Ordinary users may be so confused about all the screens that they cannot express the problems they have. This doesn't help the designer improve the user interface. Expert users that have been involved in the development tend to look for the system's ability to handle special situations, and they cannot see the problems that novices will encounter.

Number of evaluators

One at a time. Evaluators may work in isolation. Each evaluator writes down his own list of problems. With several evaluators, this gives the designer a heavy job trying to combine all the problem lists.

Combined list. Evaluators may be asked to come up with a combined list where each problem is mentioned only once. Also, problems mentioned by only one evaluator must be on the list. This gives the evaluators a heavy job reviewing each other's lists.

Common list. Evaluators may be asked to come up with a common list of problems – the problems that they all agree on as potential problems. In practice, this means that some evaluators have to realize that they have missed some problems that others detected. Other evaluators have to realize that a problem they pointed out may not be a problem after all. All of this requires time-consuming negotiations, and for this reason it is rarely done in practice.

Early evaluation

Early evaluation is highly important, and at this time only a mock-up is available. The typical heuristic approach will be as follows:

- Introduction: A developer briefly explains the screens.
- Method: Subjective judgement supplemented with heuristic rules. Screens are largely assessed one by one. (Task-based evaluation is not suitable.)
- Evaluators: Usability specialists if available. Sometimes fellow developers with a good sense of usability.
- Number of evaluators: Two. Developers combine the lists into one.

  This is the approach we assume in the first law of usability. It has a 50% hit-rate and reports 50% false problems, primarily because developers explain the screens and because evaluation is done screen by screen rather than task-wise.

14.2 Heuristic rules and guidelines

Heuristic rules can serve several purposes:

- Guide the designer during the design process (the design rules for virtual windows are examples).
- Help evaluators identify problems in the user interface (checking that the rules are followed).
- Explain observed usability problems (why did the user make this mistake).

Many authors have published heuristic rules for user interfaces. Already in 1986, Shneiderman published the 'eight golden rules' of dialogue design (also in Shneiderman 1998). The rules are as follows (with shortened explanations).

Eight golden rules of dialogue design (Shneiderman 1986)

1. Strive for consistency. (Use the same terminology and procedures in all parts of the user interface.)
2. Enable frequent users to use short cuts. (Examples are short-cut keys, abbreviations and macros.)
3. Offer informative feedback. (The system should indicate what it is doing and which state it is in.)
4. Design dialogues to yield closure. (A dialogue should give the user the sense of having completed his task.)
5. Offer simple error handling. (The system should detect errors and inform the user of how to handle the situation. As far as possible, the system should prevent the user from making errors.)
6. Permit easy reversal of actions. (Provide *Undo* as far as possible.)
7. Support internal locus of control. (Make the user feel that he is in control – not the system.)
8. Reduce short-term memory load. (Users should not have to remember information from one screen to another, they should choose from lists rather than type commands, etc.)

These rules are still valid today. Shneiderman thought of the rules as design guidelines to be used by the developer while he designs the user interface. They can of course also be used by heuristic evaluators to identify usability problems.

Around 1990, Jakob Nielsen and Rolf Molich developed a list of heuristic rules, specifically aimed at heuristic evaluation (Molich and Nielsen 1990). The authors also used the rules to explain observed problems. The list is rather similar to Shneiderman's golden rules. The list has later been modified, extended to 10 rules

and published in other places. Here is the original list, illustrated with problems encountered in the hotel system.

Molich and Nielsen (1990)

1. Simple and natural dialogue. (Avoid irrelevant information and do things in a natural and logical order.) This rule has no counterpart in Shneiderman's golden rules. It is a very broad rule that can vaguely explain many problems. In the hotel system, we might for instance claim that the rule explains why many users didn't find the menu that helped them print a booking confirmation: the menu was not in a natural order relative to the fields filled in by the user.

2. Speak the user's language. (Use words familiar to the user. Avoid computer and system jargon.) This rule too has no counterpart in Shneiderman's golden rules. In the hotel system, the term *Stay* was unfamiliar to most users. Yet most users soon guessed what it meant. Surprisingly, hotel people had no term that covered the system concept of a stay, so the problem was not easy to repair. Users sometimes said *guest*, sometimes *booking*, but none of these correctly reflected the stay concept, which users praised once they understood it.

3. Minimize the user's memory load. (Users should not have to remember information from one screen to another, they should choose from lists rather than type commands, etc.) None of the hotel system problems related to this rule, thanks to the use of combo boxes and a minimal set of windows.

4. Be consistent. (Use the same terminology and procedures in all parts of the user interface. Follow platform conventions.) None of the hotel system problems directly relate to this rule. Actually, some problems were caused by an attempt to follow the platform conventions. The Microsoft Windows guidelines said that a window had to have a *File* menu. In the hotel system there was no natural use for a File menu, so the designers twisted their mind to put something in it. The users didn't buy the solution.

5. Provide feedback. (The system should indicate what it is doing and which state it is in.) This rule can explain some of the problems in the hotel system, for instance that users couldn't see whether they had completed the check-in task. There was no visible feedback.

6. Provide clearly marked exits. (Users should always see the way back, for instance when they by mistake have entered a wrong screen.) You have to stress your imagination, however, to make the rule cover undo. The rule partly matches Shneiderman's rule of *Easy reversal of actions*, which clearly covers undo.

7. Provide short cuts. (Intended for experienced users, unseen by the novice.) The hotel system tried to follow this guideline, but in a few places it didn't. However, we couldn't observe the problem because we didn't try with experienced users. This is a good example of a guideline that exceeds what we can easily find with usability tests.

8 Provide good error messages. (Error messages should be friendly, precise and constructive.) The hotel system tried to follow this rule, but sometimes missed. One example is that the user wanted to extend a booking with one more day. In the room grid, the user marked the entire period including the nights booked already and clicked *Book*. The system replied that the room wasn't free in the entire period and suggested that the user choose another room. The user was bewildered because the room was already partially booked by the guest and free the next night. Although the system formally was right in its statement, it should have recognized that the guest had the room already.

9 Error prevention. (Prevent users from starting erroneous actions.) This rule is part of Shneiderman's rule of simple error handling. In the hotel system, the error mentioned under point 8 could have been prevented if the user was only allowed to select free rooms. In general, the use of combo boxes and lists to choose from, prevented a lot of errors. At the same time, it reduced the user's memory load.

Example: an error message from Microsoft Word

Let us try to use the heuristic rules on the situation shown in Figure 14.2. The example is from an old version of Microsoft Word. The user tries to set the line spacing so that there is half a line empty after the text. The system responds as shown in the figure, and even offers help. When the user tries the help, the result is as shown. It should be obvious that the dialogue is ridiculous and almost insulting. The problem is that the user should have typed *0.5 li* rather than *0.5 lines*. Now, what do the Molich–Nielsen rules tell us about it?

1 Simple and natural dialogue? Not quite. It is not obvious what *Before* and *After* mean. Before what?

2 Speak the user's language? Not quite. The user tries to write *lines*, the system insists on *li*.

3 Minimize the user's memory load? Even if the user figures out what to type, it is hard to remember the next time.

4 Be consistent? No problem. Or will the user miss an OK button in the Help box?

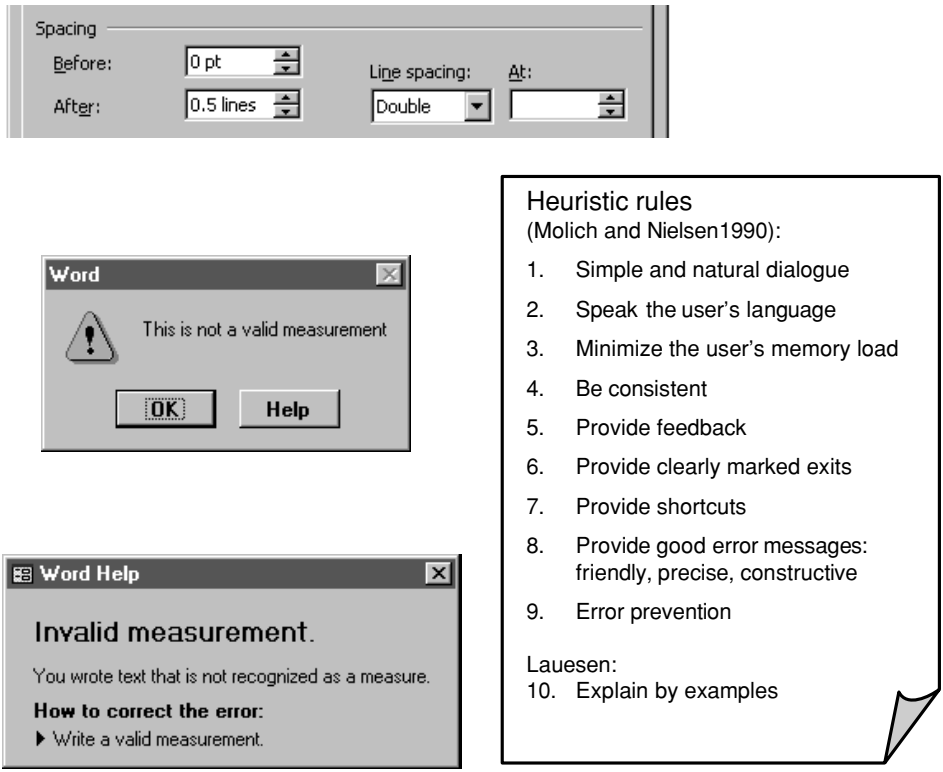
5 Provide feedback? No problem. If the user succeeds, the system will give visual feedback in the form of the new line spacing.

6 Provide clearly marked exits? No problem. Or will the user miss an OK button in the Help box?

7 Provide short cuts? No problem. There are shortcuts in the standard way.

8 Provide good error messages? (Error messages should be friendly, precise and constructive.) No, definitely not. Telling the user that he has done something invalid is not friendly. The message is not precise because it doesn't say which

Fig 14.2 Heuristic rules



field is wrong. Finally, it is definitely not constructive because it doesn't really tell the user what to do – it only pretends to do so.

9 Error prevention? No, definitely not. A solution would be to allow the user to select the measurement unit from a list.

What is the result of this rule-based evaluation? Several potential problems and suggested solutions, from explaining what *Before* and *After* mean, to adding a Close button on the Help box. As evaluators, we might report all of these problems and suggestions. Imagine that we used the heuristic rules in the same systematic way on every detail of the user interface. It would take a long time and the designer would be flooded by potential problems and suggestions.

If we look for a simple and fast solution to the line-spacing dialogue, I would use an additional heuristic rule of my own.

Lauesen

10 Explain by means of examples.

This rule would tell the designer to change the error message to, for instance:

Write the spacing as 0.5 li, 0.5 pt or 0.5 mm.

How can it happen?

This example from Word is quite scaring, but also very common. In Office 2000, the error message is still the same, except that there is no Help button in the message anymore.

Most of us often see nonsense messages and help texts similar to the one here. How can it happen? The reason is that error messages and help texts are made by technical writers – often at the last moment before release. The technical writer may master the English language but doesn't understand what the system does. Imagine that you were a technical writer. You got this message box to fill in – among a hundred others. You have to write *something* but haven't got the time to find out when the message occurs, what a *measure* is and what the user is allowed to write. Under these circumstances you end up writing something that pretends to give an answer but is sheer nonsense.

14.3 Cost comparison

Is heuristic evaluation cheaper than usability tests? Sometimes – it depends on how the techniques are carried out. Let us look at low-cost approaches early during design – the most critical point in user interface development. At this point in time, it doesn't make sense to test with a lot of users. We know that there will be serious problems that most users encounter, and we want to find them early. We may run a usability test with three users, one by one. We assume that we have developers with some usability background. They will run the tests, one as a facilitator and one as a log keeper. In the low-cost version, this should take a total of 11 work hours, including writing the test report (see details in section 13.2):

	Work hours	
Series of three users	**Facilitator**	**Log keeper**
Total for one user	80 min	80 min
Total for three users	240 min	240 min
Writing the test report		180 min
Total work hours		660 min (11 hours)

Let us compare it with a low-cost heuristic evaluation with two independent evaluators. We assume that each evaluator uses his subjective judgement and lists the problems he finds. Then he meets with a developer to explain his findings. (This approach corresponds to inspection techniques used for program reviews. The alternative is to write a detailed report. It will take much longer, and the developers will most likely misunderstand it anyway.)

The time needed for the evaluation depends on the number of screens and their complexity. We assume that the system at this stage has around 8 screens of medium complexity. The total time used will be around this.

	Work hours		
Two evaluators, subjective judgement	**Clock hours**	**Evaluators**	**Developer**
Introduction to the system	0.5	1.0	0.5
Evaluating the system	2.0	4.0	
Listing the problems	1.0	2.0	
Explaining them to developer	1.0	2.0	2.0
Total	4.5	9.0	2.5

In this case, heuristic evaluation is a bit more expensive, 11.5 hours against 11. The figures change of course, if we use fewer or more test persons and evaluators, or if we vary the reporting approach. If evaluators check against heuristic rules, time for heuristic evaluation will rise.

However, there are additional costs that we haven't mentioned. One is the time it takes to prepare a mock-up for early usability testing. Making the basic screens is much the same whether the screens are to be used for usability testing or heuristic evaluation. However, to prepare for usability testing, we have to make screen copies and fill them with realistic data. Depending on the kind of system, this can take several hours.

Another factor is the time it takes to find test persons versus heuristic evaluators. This again depends on the kind of system and the developer environment. Sometimes finding test users takes much time; in other cases it is harder to find the right kind of evaluators.

14.4 Effect comparison (CUE-4)

Early evaluation

Let us ignore the cost for a moment and look only at the effectiveness: the number of problems found. When used early in development with paper mock-ups, heuristic evaluation is less effective. It finds only around 50% of the problems that real users encounter. Also serious task failures may be overlooked by evaluators. The main reason seems to be that evaluators at this point in development cannot experiment with the system. They need a bit of introduction from the developer and tend to evaluate the system screen by screen, rather than task-wise.

Heuristic evaluation also reports a lot of false problems – problems that real users don't encounter, or only very few users. Trying to correct all of these false problems is much more costly than any time saved by using heuristic evaluation rather than usability testing.

Properly used, heuristic evaluation is valuable anyway. The trick is to consider heuristic evaluation a help similar to when you have someone read and comment on a paper you have written. Many of the comments are obvious when you hear them – why did I overlook this, the author wonders. Other comments are more dubious, and you may decide not to deal with them – particularly if it hard to do so. However, the difference between writing a paper and designing a user interface is that if a reader gets stuck, he can most likely just skip a small part of the paper. But if users get stuck, the problem is real.

Effectiveness is not only a matter of precision: finding all the true problems and no false ones. It is also about coverage – how many screens can we deal with. Here, heuristic evaluation is more effective than usability testing. While users get exhausted after about an hour and may have used only a few screens, evaluators can keep going for several hours and can cover a lot of screens in that time.

Late evaluation (CUE-4)

When the system is operational, heuristic evaluation may work more like usability testing. The evaluators can experiment with the system and use themselves as test users. We will look closer at an ambitious comparison project in this area.

At the CHI 2003 Workshop, Rolf Molich and Robin Jeffries had arranged the CUE-4 experiment (Molich 2003). They had persuaded 17 of the world's best usability teams to evaluate the same public Web site, www.hotelpenn.com. This site offers on-line booking on *New York's Hotel Pennsylvania*. It had been used daily for around a year.

Eight of the usability teams were requested to use heuristic evaluation (expert evaluation) and the remaining nine teams to use usability tests. Apart from this, the teams could choose the evaluation approach (variant) they preferred. The result of

their evaluation should be a list of the problems they had identified and the degree of seriousness (category) for each problem. They were instructed to report not more than 50 problems per team. The data is available for download (see Molich 2003).

Six teams reported 50 problems: one team 49 problems and the remaining teams between 20 and 36 problems. Apparently, the limit of 50 had an influence on what seven of the teams did. There was no apparent correlation between the number of problems and whether the teams used heuristic evaluation or usability testing.

My Masters student Mimi Pave Musgrove studied all the problem reports to see whether there was any significant difference between *E-teams* (expert/heuristic evaluation) and *U-teams* (usability test). I helped with the statistics.

Making a list of distinct problems

The first task was to make a list of distinct problems, with each problem mentioned only once. This was an immense task. First of all there were around 600 problem reports. For each of these, Musgrove had to see whether the problem was on the distinct list already or whether it had to be added to the list. This was usually quite hard since teams reported in different styles and used different terms to report essentially the same problem.

In many cases it was not clear whether two problem reports were the same problem or two different ones. As an example, when one report pointed out an unreadable black text on dark-green background in screen B, and another report pointed out a similar text in screen C, do we then have one or two distinct problems? In this case Mimi decided that it was the same problem. A careful developer would correct both problems if directed to just one of them.

In some cases a single report contained two essentially distinct problems, and in other cases a team reported essentially the same problem twice. Sometimes the problems were reported as suggestions for change rather than a problem. And sometimes a positive finding was reported, for instance *the user liked the one-screen approach*, together with a negative one from another team, for instance *the one-screen approach is annoying*.

Problem counts

The end of this hard work was a list of 145 distinct problems with an indication of the teams that had reported each problem. Figure 14.4A shows the first problems on this list. The teams were identified as A, B, C . . . In the figure they are arranged so that the E-teams are shown first, and the U-teams last. As an example, problem 3 was detected by four E-teams (teams B, D, G and R) and by six U-teams (teams A, H, K, L, N and S). In total, it was detected by 10 teams (hit-rate = 10). Problem 3 was that it was difficult to compare promotion price and normal price. Users had to look at different screens to see the two prices.

The seriousness of the problem is in most cases stated as the code P, Q or R, roughly corresponding to minor problem, medium problem and task failure. Codes A, C and T show good ideas, positive remarks and program errors (bugs). Note that different teams may report the same problem with different degrees of seriousness. Problem 3 is a good example of this. Many teams report that it is a serious problem, and many report that it is a minor problem. Team R, however, doesn't explain the problem, but gives a suggestion for a change in the user interface.

The middle part of Figure 14.4A gives an overview of the hit-rates for all 145 problems. No problem was reported by all 17 teams, but one problem was reported by 16 teams. Two problems were reported by 10 teams, one of them being problem 3 as we explained above. There were 54 singular problems – reported only once.

Compared to the similar statistics for early problems (Figure 13.4), we have few high-hit problems but still a lot of singular problems. Most of the high-hit problems had probably been removed during development.

If we ignore all reports that are not task-failures (degree R), there are only 49 problems left. Their hit-rates are shown on the lower part of Figure 14.4A. The remaining 96 problems never showed up as something serious. The top-hit was problem 25, which nine teams found serious. There are still a surprising number of singular problems among the R-problems. All of them looked serious to the teams that reported them.

The bar graphs only tell us how many times each problem was reported in total. We cannot see how many times it was reported by E-teams versus U-teams. If we look at the raw data in the list, problem 3, for instance, was reported roughly the same number of times by E- and U-teams. In contrast, problem 6 was reported once by E-teams and four times by U-teams. Does this mean that U-teams were better at detecting this particular problem? The short answer is *yes, but it is a coincidence.* We will now look closer at this question.

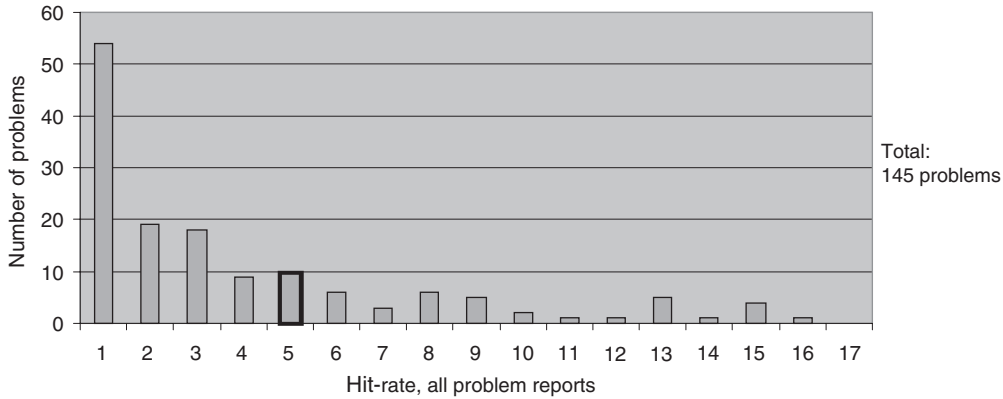
First, we need a better overview of E- versus U-problems. Figure 14.4B shows a matrix with all problems according to their E- and U-hit-rates. As an example, look at the top row of the matrix. It shows all the problems that were not reported by any E-team. Of these, 24 problems were reported by a single U-team. Seven were reported by two U-teams, another seven by three U-teams, and one problem by four U-teams.

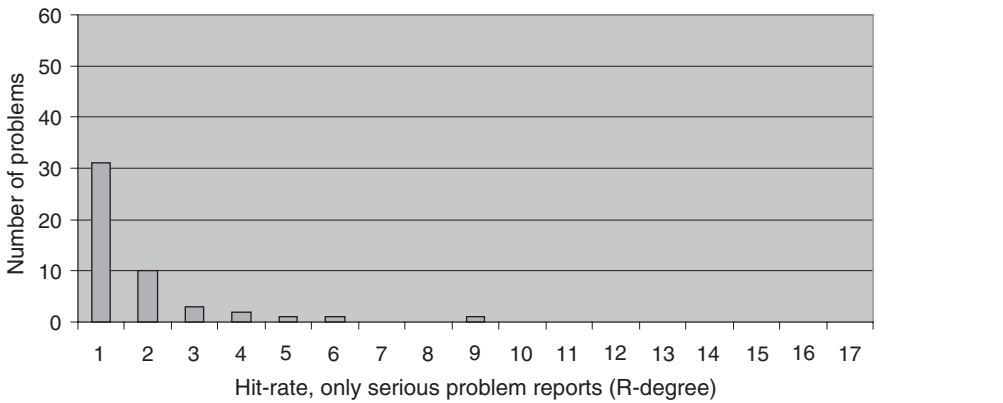
Similarly, the first column shows the problems that were not reported by any U-team. Of these, 30 problems were reported by a single E-team, three by two E-teams and one by four E-teams. All other problems were reported by E-teams as well as U-teams. As an example, three problems were observed by seven U-teams and six E-teams.

Fig 14.4 A Problems reported by 17 top-level HCI teams

CUE-4 hits per distinct problem

ProblemID	B	C	D	E	F	G	P	R	A	H	J	K	L	M	N	O	S	Hit-rate
1	R		R			Q	Q	Q	R	Q		Q	Q	Q	Q	R	R	13
2						Q	Q										R	3
3	Q		Q			R		A	Q	R		P	P		Q		Q	10
4													Q					1
5	Q			Q		R			R							Q		5
6		Q							P	R			P	Q				5
7				C	R		Q	R	Q	P	Q	C	Q	R	P	Q	Q	13
8						R			Q								R	4
9		Q	R	R		R	Q	R	P	Q		Q	Q	R	R		Q	13
			E-teams									U-teams						





In summary, we have these figures:

	Count	Percentage of U-problems
Problems observed by E but not by U	34	31% (false problems)
Problems observed by U but not by E	39	35% (missed problems)
Problems observed by E as well as U	72	
Problems in total	145	

These figures are somewhat better than predicted by the first 'law' of usability. The first law claims that heuristic evaluation misses 50% of the problems detected by usability tests. In our case heuristic evaluation missed only 35%. The first law also claims that heuristic evaluation predicts 50% false problems. In our case it predicted only 31% false problems. And even so, are these 'mistakes' really mistakes or only coincidences? In order to answer this question we need a statistical analysis.

Statistical analysis

If E- and U-teams are equally good at detecting problems, the number of E-hits and the number of U-hits would be almost the same for a given problem. As a result, we would expect the problems in Figure 14.4B to lie in the white band around the diagonal from top-left to bottom-right. To some extent they do so, but it is not too clear.

Let us state our expectation as a statistical hypothesis:

Hypothesis A: E- and U-teams detect a given problem with the same probability

What would this mean in practice? Assume that we have a problem that is observed five times because of its nature. The middle of Figure 14.4B shows these five observations as five balls. According to the hypothesis, they fall at random into the 17 bowls representing the 17 teams. However, in our case each team can receive at most one ball because they were asked to report the problem at most once.

What are the possible outcomes of this? Well, the U-teams can together get either zero, one, two, three, four or five balls. The E-teams will get the remaining of the five balls. We can compute the exact probability of each of these outcomes, given our hypothesis. The figure shows these probabilities. As an example, the chance of four U-teams getting a ball is 16.3%. This is exactly what happened to problem 6 above. It was observed five times, four of them by U-teams. So this outcome is quite probable.

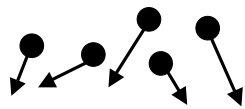
Note that the probabilities are not symmetrical. The chance of zero U-hits is smaller than the chance of zero E-hits. This is because there are more U-teams, so the chance of them getting nothing is smaller.

Fig 14.4 B Problems according to E-hits and U-hits

All problems

E-hits \ U-hits	0	1	2	3	4	5	6	7	8	9
0		24	7	7	1					
1	30	9	6	3	2	2				
2	3	5	1	3	2	1				
3		3	4	1	1	3	2		1	
4	1	1	1	1		2	2			1
5				3	1				1	
6								3		2
7								1	2	1
8					1					

Five observations dropped at random



B	C	D	E	F	G	P	R	A	H	J	K	L	M	N	O	S

E-teams U-teams

U-hits:	0	1	2	3	4	5
E-hits	5	4	3	2	1	0
Probability:	0.9%	10.2%	32.6%	38.0%	16.3%	2.0%

Probability of a problem observed N times having
E expert observations and U usability observations

E-hits \ U-hits	0	1	2	3	4	5	6	7	8	9
0		52.9%	26.5%	12.4%	5.3%	2.0%	0.7%	0.2%	0.0%	0.0%
1	47.1%	52.9%	42.4%	28.2%	16.3%	8.1%	3.5%	1.2%	0.3%	0.0%
2	20.6%	37.1%	42.4%	38.0%	28.5%	18.1%	9.7%	4.1%	1.3%	0.2%
3	8.2%	21.2%	32.6%	38.0%	36.3%	29.0%	19.4%	10.4%	4.1%	0.9%
4	2.9%	10.2%	20.4%	30.2%	36.3%	36.3%	30.2%	20.4%	10.2%	2.9%
5	0.9%	4.1%	10.4%	19.4%	29.0%	36.3%	38.0%	32.6%	21.2%	8.2%
6	0.2%	1.3%	4.1%	9.7%	18.1%	28.5%	38.0%	42.4%	37.1%	20.6%
7	0.0%	0.3%	1.2%	3.5%	8.1%	16.3%	28.2%	42.4%	52.9%	47.1%
8	0.0%	0.0%	0.2%	0.7%	2.0%	5.3%	12.4%	26.5%	52.9%	100.0%

How do we compute the probabilities? In principle, we compute the total number of ways that five balls can fall into 17 bowls. Then we compute how many of these ways give zero U-balls, one U-ball, etc. The probability is the number of 'good' ways divided by the total number of ways.

Technically speaking, the probabilities are a hypergeometric distribution. In Microsoft Excel, we can compute the probability of getting x out of five balls with this formula:

> Probability(x U-hits) = HypGeomDist(x, 9, 5, 9+8)
> when there are 9 U-teams, 8 E-teams and 5 hits in total

In the same way, we can compute the probabilities when there are 6, 7 and other hits in total. In the lower part of Figure 14.4B, we have shown these computed probabilities in a matrix. You find our 5-ball probabilities in the skew band of framed cells. The circle shows where problem 6 belongs.

We have coloured all cells that are unlikely as outcome. The colour is dark if there is less than 5% probability of a problem ending up here or in cells further away from the diagonal. The colour is light if there is between 5 and 10% probability of a problem ending up here or further away from the diagonal.

As an example, a 5-hit problem with zero E-hits has a probability of 2% and is thus in dark colour. A 6-hit problem with one E-hit has a probability of 8.1%. With zero E-hits it has a probability of 0.7%. In total, it has an 8.8% probability of being this far away from the diagonal. Thus, it has a light colour.

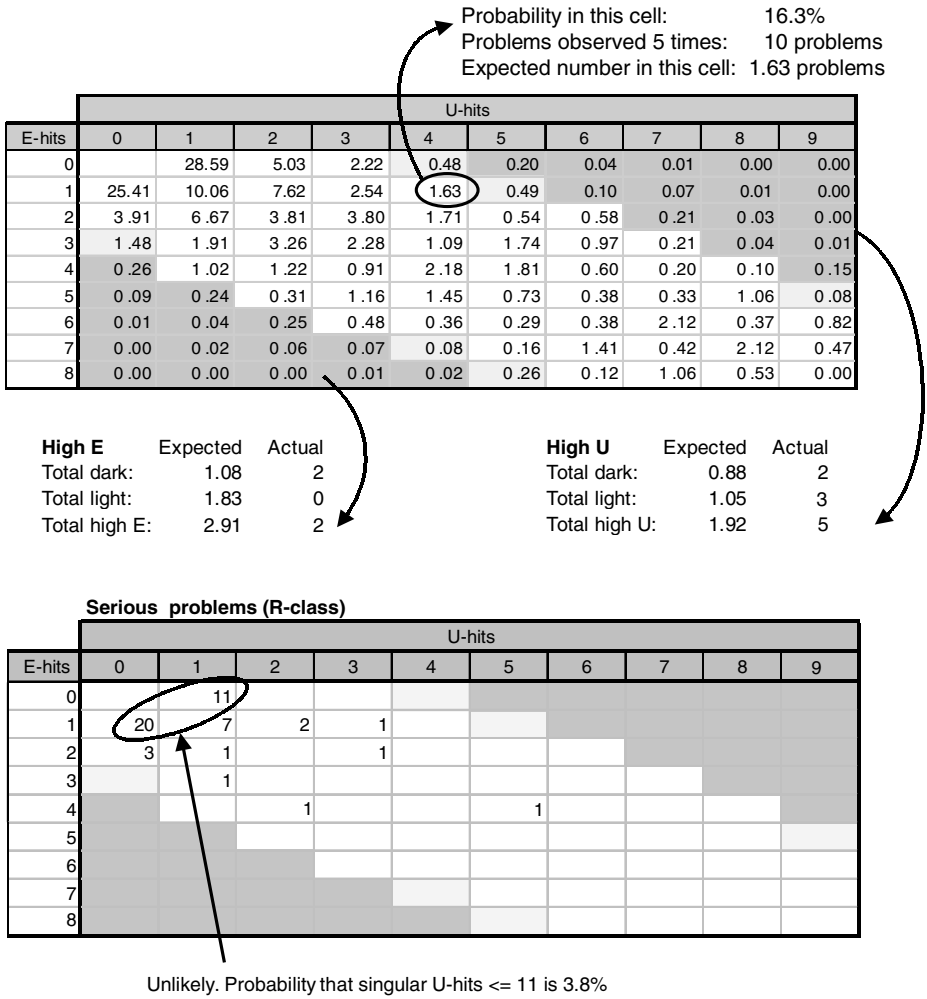
These light and dark colours are the ones you also see at the top of the figure. Notice that there are a few problems in the unlikely, coloured areas. In the upper right-hand area there are five unlikely problems. In the lower left area there are two unlikely problems. All other problems are in very likely positions.

Can we say that the odd problems are unlikely and that they prove a difference between E- and U-teams? No, because the unlikely, coloured areas cover cells with probabilities below 10%. Some problems should by chance end up there. Our first guess might be that 10% of all problems should end up in the upper right-hand area. We would thus expect 14.5 problems there. There are only five! Something must be wrong. Maybe some teams have made a secret agreement on sharing the problems more evenly than chance?

Fortunately, our guess of 10% was wrong. It would be right if the outcomes had been distributed over a continuous scale, for instance according to a normal distribution, but in our case they are not. Our observations are only yes or no, and we count the yeses.

Figure 14.4C shows a precise calculation of the expected number of 'unlikely' problems. Let us as an example look at the circled cell with four U-hits and one E-hit. A problem observed five times would end up in this cell with a probability of 16.3%.

Fig 14.4 C Observed problems: predicted hits on average



There are ten 5-hit problems in the data. As a result we would expect 16.3% of the ten problems to end up here, in other words 1.63 problems on 'average'.

The figure shows the expected problems in every cell. We have computed the total expected problems in the unlikely areas. For instance in the dark-coloured high-U area, we would expect 0.88 problems. Actually two problems are observed in this area. In the entire high-U area we would expect 1.92 problems, but actually there are 5. There is a small difference.

Similarly, in the high-E area we would expect 2.91 problems, and 2 are observed. We cannot get much closer.

Conclusion A

Our hypothesis was that E- and U-teams detected a given problem with the same probability. This has been confirmed with high accuracy. Maybe U-teams are slightly better at finding problems, but they have at most done it for two out of the five high U-problems. (We cannot say which of these five.)

Does this disprove the first law of usability, that E-teams should find only 50% of the U-team problems and produce 50% false problems? Yes it does, but only for the conditions that this project dealt with:

- Late evaluations where E-teams can experiment with the product and use themselves as test subjects
- Top-level HCI people
- Very few high-hit problems
- A dedicated effort by U-teams as well as E-teams to find unusual domain situations

The last condition is not documented in the numbers involved, but it is obvious when you read the test reports. It means that creativity in identifying domain situations has a large influence on the problems reported. This creativity is much the same for all top-level HCI teams, whether they use heuristic evaluation or not.

Narrow down to serious problems

The analysis above dealt with all problems. Are there differences when we look at only the serious problems (degree R)? The bottom part of Figure 14.4C shows the E–U matrix when we only include problem reports of degree R. The singular problems dominate and there are very few other problems. No problem at all is in the unlikely areas. There are simply too few problems for any to hit the unlikely cells.

However, there is a suspicious thing in the matrix. The singular problems are wryly distributed: 11 for the U-teams and 20 for the E-teams. In the first matrix they were more evenly distributed: 24 U-reports and 30 E-reports. Are these distributions unlikely?

Let us test this statistical hypothesis:

> **Hypothesis B:** E- and U-teams detect a given singular problem with the same probability

This hypothesis is only slightly different from the first one, but the statistical model is different because each problem is observed only once. We now imagine each singular

problem as a ball being dropped at random into one of the bowls – meaning that it is observed by one of the teams. Contrary to our first hypothesis, each bowl may in this case end up with several balls.

Since 9 of the 17 teams are U-teams, a ball will end up in a U-bowl with a probability of 9/17, assuming that our hypothesis is true. We can compute the exact probability that the U-bowls end up with x out of N balls. Technically speaking we have a binary distribution with $p = 9/17$. In Excel, we can compute the probability with this formula:

Probability(x singular U-problems) = BinomDist(x, N, 9/17, False)

when there are N singular problems in total, 9 U-teams and 17 teams in total

In our case, N is 31. The probability of x being 11 comes out as 2.2%. We want to find the probability that $x \leq 11$. We can do this by computing the probabilities for $x = 11, 10, 9, \ldots, 0$ and adding them. Excel can do this for us when we let the last parameter be *True*. The result is

Probability($x \leq 11$) = 3.8% when we look only at serious, singular problems

Or as the statisticians say, this wryness is significant on the 3.8% level. When we compute the same for all problems – not just the serious ones – we get

Probability($x \leq 24$) = 13.3% when we look at all singular problems

According to statistical tradition, this would not be considered a significant wryness.

Conclusion B

E-teams report significantly more serious, singular problems than U-teams. If we look at all the singular problems, there is no significant difference between U and E. The reason seems to be that heuristic evaluators judge the singular problems as more serious.

This is not so surprising after all. It is hard for an expert to judge how serious a usability problem is to an ordinary user. It is more surprising that a similar difference doesn't show up for non-singular problems. Unfortunately, there are much too few serious problems to reveal any difference.