Extracted from:

Designed for Use, Second Edition

The Pragmatic Bookshelf

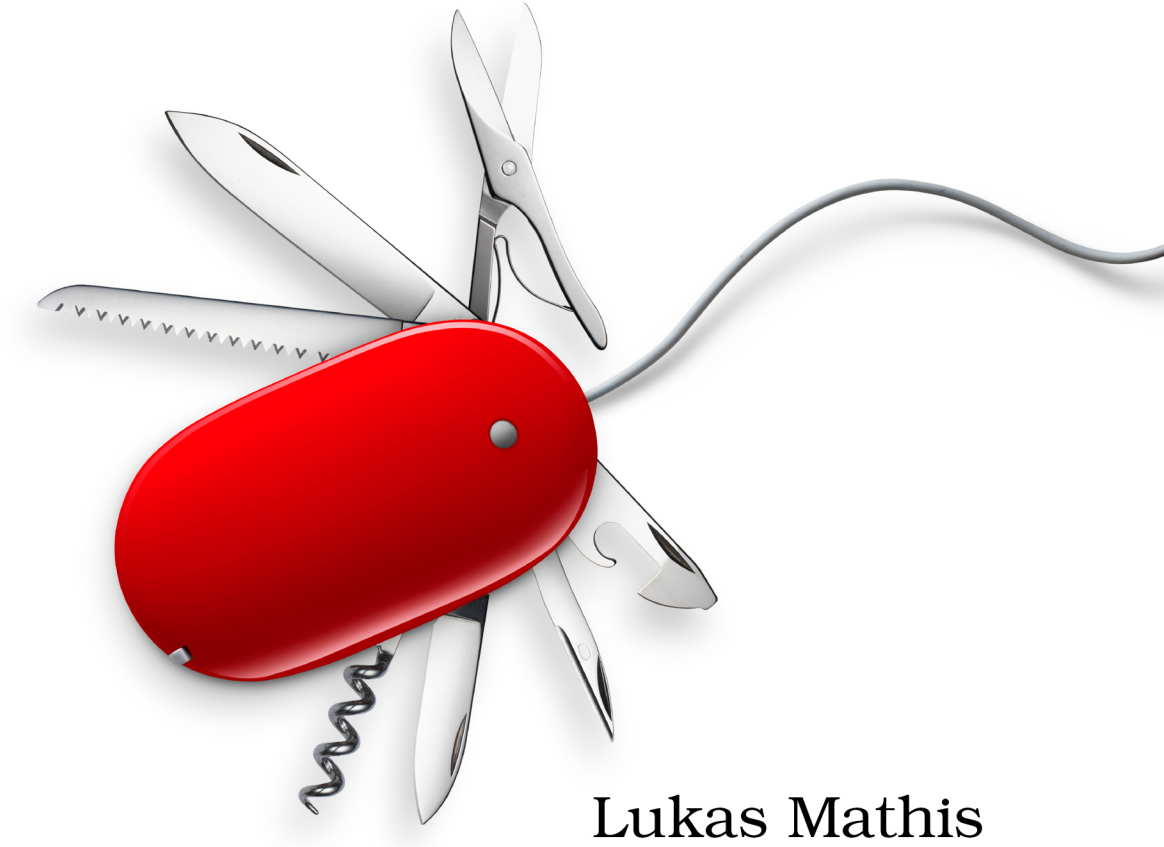
Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

Second Edition

Designed for Use

Create Usable Interfaces for Applications and the Web

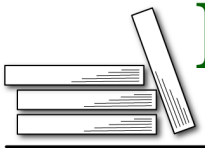


Lukas Mathis

Designed for Use, Second Edition

Lukas Mathis

The Pragmatic Bookshelf
Dallas, Texas • Raleigh, North Carolina



Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *https://pragprog.com*.

The team that produced this book includes:

Michael Swaine (editor)
Potomac Indexing, LLC (index)
Nicole Abramowitz (copyedit)
Dave Thomas (layout)
Janet Furlow (producer)

For customer support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

Printed in the United States of America.
ISBN-13: 978-1-68050-160-5
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—April 2016

For Regula and Werner

CHAPTER 36

How Not to Test: Common Mistakes

Usability tests are surprisingly resilient to mistakes. No matter how poorly you do them, chances are that you will get *some* useful information out of them. You might not uncover the most important problems, and maybe you'll get only a small subset of all problems, but you'll get *something* that will help you improve your product.

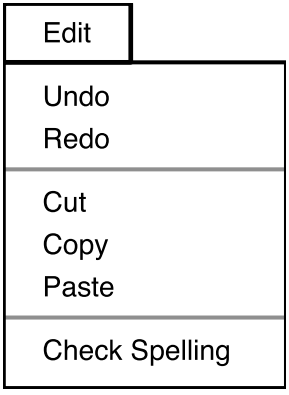
Still, if you take the time to test properly, you will get better results.

I've already mentioned some of these points in earlier chapters, but I think it's useful to do a quick recap here.

Don't Use Words That Appear in the User Interface

It's important not to use your application's terminology when you create tasks and when you talk to your tester. You want to know whether people are capable of using your application. You don't want to know whether they are able to find a specific word in your user interface.

Let's say you're testing a word processor, and you want to see whether people can use its spell-checker. The wireframe shows how this feature might be accessed in one of your application's menus.



Now, if you were to phrase the task along the lines of "Check your document's spelling," your tester could simply look through your application's menus and find the one with the matching words. It's best to avoid describing the task at all; instead, describe the goal: *"Make sure that there are no typos in your text."*

Tasks phrased like this are more aligned with what your users' goals might actually be, so the tester's behavior is more along the lines of how a real user might behave.

Don't Influence the Tester

Wilhelm von Osten was a German math teacher who lived around the early 1900s. As a hobby, he trained horses. You can probably see where this story is going: he tried to teach his horse how to do math. To everybody's astonishment, the horse (aptly named Clever Hans) quickly learned to do a number of reasonably complex math calculations: additions, subtractions, multiplications, divisions, and date calculations. The horse was capable of understanding math questions asked in plain German, and it could also read math questions if they were written on a piece of paper.

Obviously, the horse couldn't write down the results. Instead, it tapped the correct number with its hoof.

Psychologist Oskar Pfungst became suspicious and decided to investigate Clever Hans. Quite quickly, he was able to establish what was happening: Clever Hans didn't actually do any calculations. The animal didn't read, and it didn't understand German. Instead, it responded to involuntary cues in the body language of its trainer, von Osten, who, in turn, was the one who solved the math problems. The horse simply watched him and tapped its hoof until von Osten would indicate that it had reached the correct number.

Interestingly, von Osten was completely unaware that he was providing these cues to the horse.

This piqued Pfungst's interest, and he continued his experiments on the topic. He was eventually able to show that the same kind of interaction occurs between humans and, furthermore, that it is impossible to suppress these involuntary cues, even if we are completely aware of their existence.

What does this mean for us?

It means it is easily possible for a facilitator to involuntarily lead the tester through the whole test, thereby utterly invalidating the test. This is why formal usability tests are always done behind two-way mirrors; this eliminates any possibility of outside influence on the tester.

There are a few things you can do to improve the situation. First, even though you can't consciously avoid giving involuntary cues, you *can* cut down on your voluntary cues. So, don't interact with the tester unless it's absolutely necessary.

Second, you have to remove yourself from the tester's field of vision, sitting a bit behind the tester so he is less aware of your presence.

Third, if possible, run a test that doesn't require sitting with the tester. Remote testing is perfect for this (see Chapter 35, *Remote Testing*, on page ?).

Finally, don't worry about it too much; just be attentive. You are not doing a statistically valid double-blind study here; you are merely trying to find problems in your user interface. If the tester hesitates or looks to you for guidance, you already know that there might be a problem with the user interface, even if your reaction allows her to figure out what to do next.

Avoid Stressful Situations

All medical students are taught this principal precept of medical ethics: "Primum non nocere," or, "First, do no harm." The same should apply to usability professionals. When we run usability tests, we put people in a situation where we want them to make mistakes. But making mistakes can be a stressful situation; nobody likes to make mistakes. You can do a number of things to make testers feel OK with the situation they find themselves in.

First, you need to make sure you explicitly and clearly point out that you are not testing them, but the user interface.

Second, if a tester gets stuck and it becomes obvious that he's on the wrong track or becoming agitated, intervene. Say something to make it clear that the tester is not to blame, something along the lines of, "This is exactly why you're running these tests. This design still has some serious issues here. I think you'll have to go back to the drawing board on this one." Then, move on to the next task.

Third, do not repeatedly tell testers to speak out loud. It's OK to make this point at the beginning of the test by saying something like, "Feel free to speak out loud during the test; this helps me better understand where problems in your user interface are." It's also OK to ask something like "What's on your mind right now?" when it's not clear what the tester is doing during the test. But keep in mind that some people just aren't comfortable with what amounts to talking to themselves.

Fourth, if it appears that a tester is starting to get stressed out by the experience, stop the test. Besides being at risk of putting the tester through a horrible experience, you won't get any value out of forcing a tester who is upset and unable to concentrate to finish the test. His behavior typically won't

be representative of your real users, who have the ability to take a break or work on something else if your product is annoying them.

Finally, always end tests on a positive note. Thank people for their time. Tell them that their participation in the test was valuable to you and will help you improve your product.

Takeaway Points

- When designing tasks or talking to users during usability tests, avoid using words that appear in the user interface.
- Avoid influencing the tester during tests.
- Watch out for situations where the tester looks to you for guidance, since these indicate usability problems.
- Avoid putting the tester through stressful situations.
- Don't constantly ask the tester to speak out loud.
- Stop the test if the tester seems to get upset or annoyed.