1.3 Tools for Evaluating Usability: Heuristics and Think–Aloud Testing

Assessments

1.3.1 Basic Heuristic Evaluation

Heuristic Evaluation

Heuristic evaluation (HE) is a technique for analyzing the usability of an interface design at early stages of development. It was developed by researchers in Denmark a decade ago (Molich and Nielsen, 1990) to create an informal, tractable, and teachable way to look at an interface design and form an opinion about what is good and bad about it. Research has continued on this technique; the heuristics have been refined; and it is now a fairly well–established technique for user interface design (Nielsen, 1993).

HE's *heuristics*, or usability principles, have been established from the practice of UI design and evaluation. That is, they embody a compilation of good design practices and known design failures that have arisen in the UI design field over the last 30 years. They are not *derived* from the information–processing psychological theory we introduced earlier, though they can be understood in terms of that theory. In this overview, we will give a brief description of how the technique is used and a description of each of the heuristics and their relations to the information–processing theory. Examples of each of the heuristics will be presented in detail as they arise in the rest of the course.

Overview of Procedure for Using Heuristics

The basic procedure for doing a heuristic evaluation is to come up with a UI design and have several people examine that design with respect to whether it violates any of the heuristics. If it violates a heuristic, fix the design so that it doesn't violate the heuristic.

The design can be at any stage in the development of the product. It can be as preliminary as paper and pencil sketches of what the system might look like; it can be a partially operational prototype written, say, in Visual Basic; or it can be a fully operational system. However, the earlier that usability problems are found, the cheaper it is to fix them—so HE is probably most valuable when it is used on very early sketches or prototypes.

Typically, several evaluators will do a HE review of a design because research has shown that different people will find different violations. For a very simple system (like an ATM that allows people to do only a half a dozen functions), perhaps four or five evaluators are sufficient to find most of the usability problems (Nielsen, 1993). For more complicated systems, more evaluators may be necessary (Slavkovic and Cross, 1999). However, since this is such a simple technique to learn and use, it is not difficult to find development team

members who will perform this function. In addition, even one or two people doing the evaluation will find some usability problems that, if fixed, would increase the usability of the system you are designing.

Once you understand the heuristics, then you will find that you use them implicitly as you design. Therefore, these heuristics can inspire some of the details of a UI design (for example, what words to use as button labels or menus) as well as serve as an evaluation technique to use after a design has been conceived.

The following are nine heuristics listed in Nielsen (1993) and one (the last one) is from Nielsen and Mack (1994).

Visibility of System Status

Brief explanation of the heuristic. The computer should keep the user informed about what is going on; what input it, the computer, has received; what processing it is currently doing; and what the results of processing are.

Relationship to the information–processing model of a user. The computer has to supply information to the user through sight or sound; otherwise, the user will not have the information necessary to understand what is happening.

Match Between System and the Real World

Brief explanation of the heuristic. The UI design should use concepts, language, and real–world conventions that are familiar to the user. This means that the developers need to understand the task the system is going to perform—*from the point of view of the user*. This also makes cultural issues relevant for the design of systems that are expected to be used globally.

Relationship to the information–processing model of a user. Familiar concepts, language, and real–world conventions in the interface can make use of knowledge already in a user's long–term memory because of their experience with the task domain (independent of the computer).

User Control and Freedom

Brief explanation of the heuristic. Allow the user to have control of the interaction. Users should be able to undo their actions easily, exit from any interaction quickly at any time, and not be forced into a series of actions controlled by the computer.

Relationship to the information–processing model of a user. Users will make errors, and, therefore, they will need easy ways to recover from them.

Consistency and Standards

Brief explanation of the heuristic. Information that is the *same* should *appear to be the same* (same words, icons, and positions on the screen). Information that is *different* should be *expressed differently*. This consistency should be maintained within a single application and within a platform. Developers therefore need to know platform conventions.

Relationship to the information–processing model of a user. As with the **Match Between System and the Real World** heuristic, this heuristic makes use of a user's prior knowledge and experience with other parts of the same application as well as with other applications on the same platform.

Error Prevention

Brief explanation of the heuristic. As much as possible, prevent errors from happening in the first place. For instance, if there are a limited number of legal actions at some point in the application, then help users select from among these legal actions—rather than allowing them to perform any action and telling them after the fact when they've made an error. This could be considered a subset of the first heuristic, **visibility of system status** (what input the computer system is ready to accept), but it is so important and so often violated that it warrants its own heuristic.

Relationship to the information–processing model of a user. Errors can come about because users make mistakes in perception, lack knowledge about what to do next, recall the gist of a command rather than the exact details, or slip when they type or point. Some of these mistakes can be prevented by showing only those actions that are acceptable at that particular point in an interaction (e.g., graying out inappropriate buttons) or caught as soon as the user performs them (e.g., not accepting an incorrect abbreviation of a US state in an address form).

Recognition Rather Than Recall

Brief explanation of the heuristic. Show all objects and actions available to the user. Do not require them to remember information from one screen of the application to another.

Relationship to the information–processing model of a user. This heuristic is a direct application of theories of human memory. It is much easier for someone to recognize that they know what to do if there are cues in the environment coming into working memory through perception as well as knowledge in long–term memory of what to do.

Flexibility and Efficiency of Use

Brief explanation of the heuristic. The design should have accelerators (keyboard shortcuts) to allow skilled users to speed up their interaction (as opposed to always using menus or icons with the mouse). Skilled users should also be able to tailor their interface to speed up frequent actions.

Relationship to the information–processing model of a user. The value of accelerators comes primarily from the motor processes of the user. Typing single keys is typically faster than continually switching the hand between the keyboard and the mouse and pointing to things on the screen. Furthermore, skilled users will develop plans of action, which they will want to execute frequently, so tailoring can capture these plans in the interface itself.

Aesthetics and Minimalist Design

Brief explanation of the heuristic. Eliminate irrelevant screen clutter.

Relationship to the information–processing model of a user. This heuristic relates to the visual search aspect of perception and also to memory. The more clutter, the more information the eyes must search through to find the desired information. In addition, the more information coming in through perception as the visual search proceeds, the more this information interferes with the retrieval from long–term memory of the information that is actually relevant to the task at hand.

Help Users Recognize, Diagnose, and Recover from Errors

Brief explanation of the heuristic. Error messages should be written in plain language, tell the user what the problem is, and give constructive advice about how to recover from the error. Again, this could be considered a subset of the first heuristic, **visibility of system status**, but it is so important and so often violated that it warrants its own heuristic.

Relationship to the information–processing model of a user. This is simply an admonition to give the user sufficient information to understand the situation.

Help and Documentation

Brief explanation of the heuristic. If the system is not an extremely simple, walk–up–and–use application, it is going to need help and documentation. This should be always available, easily searchable, and give concrete advice applicable to users' tasks.

Relationship to the information–processing model of a user. This is again an admonition to give the user sufficient information to understand the application. The search feature should allow information to be found by asking for the gist of the meaning, rather than only the exactly right keyword, because people will remember only the gist, not the exact words (if they ever knew them).

Summary

If you keep these ten heuristics in mind as you design a user interface, your system will be more usable than it will if your design violates them. As early as possible (e.g., with sketches or a prototype), have several people examine the UI design using these heuristics to discover violations and give you plenty of time to fix them.

As we progress in the course, we will repeat these heuristics, giving concrete examples of violations and how to address them.

References

Molich, R., and Jakob Nielsen. "Improving a Human–Computer Dialog." *Communications of the ACM* 33, no. 3 (1990): 338–48.

Nielsen, Jakob. *Usability Engineering*. Boston, MA: Academic Press (AP Professional), 1993.

Nielsen, Jakob, and Robert L. Mack, eds. *Usability Inspection Methods*. New York, NY: Wiley, 1994.

Slavkovic, A., and K. Cross. "Novice Heuristic Evaluations of a Complex Interface." *Proceedings Companion of CHI 1999 (Pittsburgh, Pennsylvania, May 15–20)*. New York, NY: ACM, 1999.

1.3.2 Basic Think–Aloud Usability Testing

- References

Think–aloud usability testing is an empirical technique for assessing the usability of a prototype of an interface. This technique "may be the single most valuable usability engineering method" (Nielsen, 1993, p. 195). The essence of the technique is for you to ask a user to think aloud while performing a task on your system; you watch silently and learn how the user thinks about the task and where the user has problems using your system.

Although it is possible to do a think–aloud test with a paper prototype, it is much easier and more realistic to use a running prototype, so this technique is usually performed after a prototype is built.

Heuristic evaluation assumes that an analyst, often a developer of the system, can apply a few design heuristics successfully to predict usability problems. Several of these heuristics implicitly require that the analyst think "like a user" (e.g., **match between system and real world** requires the analyst to know the task domain of the user and "speak the user's language"; **consistency and standards** requires the analyst to know what the user considers consistent and what other standard interfaces they already know.) To the extent that a developer can do this, the heuristic evaluation technique will give good predictions. On the other hand, think–aloud usability testing assumes that the analyst can no longer "think like a user" because the analyst knows too much about the system (and sometimes too little about the task domain). Therefore, an empirical technique in which representative users actually interact with a prototype is the best way to discover what a typical user might think or do with the system. These two techniques will discover different usability problems, so we recommend that both techniques be used in designing a system.

The procedure for doing an effective think–aloud usability test is relatively simple. First, typical tasks are created that typical users might do with your system. Then users are identified who represent the types of people you expect to use your system; they should have similar educational backgrounds and computer experience to your intended users. These people are invited into a lab, are informed about the purpose and procedure of the test, and then are trained to think aloud. They are then asked to think aloud while doing the prepared tasks. Typically, they are videotaped while they work. They are allowed to use the system in any way that they can to accomplish the task without interruption or help. Afterwards, the videotapes are analyzed for "critical incidents" (described in detail in a later module) that are clues to what is problematic and needs to be fixed in the next version of the system, or to what is easy to use and should be preserved in the next version.

The basic psychological concept behind think–aloud testing is that users can voice their thoughts; specifically, they can speak aloud the contents of their working memory as they work on a task (Ericsson and Simon, 1993). Speaking aloud the contents of working memory is easiest to do with the linguistic contents of working memory (such as when they are searching a menu for a specific item), and it doesn't interfere much with the performance of the task. It is possible for people to voice the nonlinguistic contents of working memory (like searching for a specific blue in a color–selection control panel), but doing so typically slows down their performance of the task. Typically, people cannot put voice to the processing they are doing on the contents of working memory, that is, they mention the concepts or objects they are working on but not the rapid cognitive processes they are performing.

Since the users voice the contents of their working memory, interface designers who listen to their thoughts can learn a lot about what the users are thinking. For example, interface cesigners can learn which commands or which menu items make sense to users and how those commands or menu items connect to the users' task domain. They can learn which items the user sees and attends to and which he or she misses entirely. They

can learn which procedures in the interface are easy to figure out and which are totally misunderstood. They can see how the interface leads a user down a wrong path, how the user recognizes an error and backs up, how the interface helps or hinders exploration. Thus, asking a user to think aloud and then just watching the interaction—without "helping" the user—is a very valuable way to learn a lot about the usability of your system.

References

Nielsen, J. *Usability Engineering*. Boston, MA: Academic Press (AP Professional), 1993.

Ericsson, K.A., and H.A Simon. *Protocol Analysis: Verbal Reports as Data, Revised Edition*. Cambridge, MA: MIT Press, 1993.

1.3.3 How to Write a Usability Aspect Report (UAR)

Usability Aspect Reports

As you examine an interface with the techniques this course will teach you, you will find aspects of the interface that are problems for users (which you will want to fix in the next version) and aspects that are very helpful to users (which you will want to preserve in the next version). Writing a clear, useful report of these aspects (called a **usability aspect report** or **UAR**) will help to make that next version more usable. In the real world, you will write these reports for other members of your development team who have not seen the usability issue; therefore, your reports must be clear and complete. Even when you are writing these UARs just for yourself, clarity and completeness will help you understand each report six months after you write it, when you finally get around to implementing the changes it suggests.

The Elements of a UAR Report

We advocate a standard form for the report so you remember to include specific pieces of information for each usability aspect. The UAR should include the following slots:

- UAR Identifier — <Problem or Good Feature>
- Succinct description of the usability aspect
- Evidence for the aspect
- Explanation of the aspect
- Severity of the problem or benefit of the good feature
- Possible solution and potential trade–offs (if the aspect is a problem)
- Relationship to other usability aspects (if any)

We will describe each slot below—that is, what it is and why this information is necessary. We will give many examples of UARs as we introduce the details of the HE and think–aloud techniques.

UAR Identifier

This should be a unique identifier for the purposes of filing. If more than one person is working on the project or more than one analysis technique is being used, this identifier could contain letters and numbers. For example, if Chris Smith and Jan Koo are both doing an analysis, the identifier might be CS1 or JK75. If both a heuristic evaluation and a think–aloud usability study were used, the identifiers might be HE6 or TA89.

Follow the unique identifier with the word "Problem," if the report pertains to a usability problem of the interface, or the words "Good Feature," if it describes an aspect of the interface you feel should be preserved in any redesign.

Succinct Description of the Usability Aspect

This description will be used as the "name" of this UAR when you talk about its relation to other UARs. Make the name as short as possible (about three to five words) but still descriptive and distinguishable from other aspects of the system.

If this UAR is about a problem (as opposed to a good feature), make sure you have a name that describes the problem, rather than a solution. For instance, if there is a button that looks like this

Figure 1: A button with a small label.

and you think the label is too small for the average person to read comfortably, you should call the UAR "Press–Me label too small" (which is a problem statement) as opposed to "Press–Me label should be 24 point" (which is a solution, not a problem). The reason you want the name to be the problem, not the solution at this point, is that you want to leave room for the possibility that you might find several problems that are similar and that suggest one common solution. But, if you solve each individual problem individually and enshrine its individual solution in the name of its UAR, you may not see the similarities in the problems.

Evidence for the Aspect

This is the *objective supporting material* that justifies your identifying the aspect as worthy of report. This section needs to contain enough information for a reader of this UAR to understand what triggered the report. For an HE report, for instance, this could be an image of a cluttered screen and the heuristic about aesthetics and minimalist design. Or, it could be a list of menu items that do not have keyboard shortcuts and the heuristic about providing shortcuts. In a think–aloud study this is usually what was on the screen (a screen shot or description), what the user did (keystrokes, mouse movements), what the system did in response to any user actions, and what the user said. If you have video annotating or editing capability, it can be a brief animation.

You need to include enough pertinent information about the identification of an aspect for the reader to understand what the analyst was thinking when the aspect was identified (for HE) or what the user was trying to do when the aspect either hindered or facilitated the user's progress.

Explanation of the Aspect

This is your *interpretation* of the evidence. That is, for a think–aloud usability test, why you think what happened, happened, or, for an HE, why you think the aspect was designed the way it was. This can include things like "the button label, XYZ, is a standard programming term, but the user didn't seem to know that term" or "the system was in editing mode, but the user thought it was in run mode because there isn't a noticeable difference between the modes on the screen." (This should be written in a tone that analyzes what happened with the system aspect, NOT in a tone that suggests an evaluation of the developers or of the user.)

You need to provide enough context in this explanation for the reader to understand the problem—even if they do not know the system or domain as well as you do. (It is our experience that you will yourself need a lot of context when you look at these reports months in the future.)

Severity of the Problem or Benefit of the Good Feature

This is your reasoning about how important it is to either fix this problem or preserve this good feature. This includes how frequently the users will experience this aspect, whether they are likely to learn how it works, whether it will affect new users, casual users, experienced users, etc.

Possible Solutions and Potential Trade–offs

If this aspect is a *problem* (as opposed to a good feature to be preserved in the next version of the software), this is the place to propose a solution. It is not necessary to have a solution as soon as you identify a problem—you might find after analyzing the whole interface that many problems are related and can all be fixed by making a single broad change instead of making several small changes.

However, if you do propose a possible solution, report any potential design trade–offs that you see. For instance, the problem might be that there are no keyboard shortcuts for items on a menu in a mail system and you propose CTRL–S for SEND. A design trade–off you should record is that CTRL–S might already be used for another menu item (e.g., SAVE), so all shortcut keys need to be examined before any design changes are made.

Relationship to Other Usability Aspects

It is often the case that UARs are related to each other. This is where you record which UARs this one is related to and a statement about how it is related. Make sure that all the related UARs point to each other. That is, if UAR #1 related to UAR #30, then UAR #30 should point to UAR #1 in this section **and** UAR #1 should point to UAR #30 in its version of this section. (It is a common mistake to enter the pointer into a newly created UAR, but neglect to go back to the previous ones that it relates to and update their UARs.)

IMPORTANT: Always Step Back and Try to See the Bigger Picture!

This last slot in the UAR records the results of a very important step in the usability analysis process: stepping back and looking for patterns in the usability problems. You should do this at several times during the analysis. When each UAR is created, you should look back to the previous UARs and see if they are related to the new one. When you have completed all UARs, you should go back and look again for patterns. This step allows you to detect larger problems in the interface that might have a solution that fixes many small problems with only one change. This step also provides the opportunity to amass evidence for a fundamental change in the proposed interface, something that would never arise out of considering single UARs in isolation.