Mirador, Zander Miguel C.
BSCS4A

1. Starting with some of the principles outlined in Chapter 4, provide a usability specification for an electronic meetings diary or calendar. First identify some of the tasks that would be performed by a user trying to keep track of future meetings, and then complete the usability specification assuming that the electronic system will be replacing a paper-based system. What assumptions do you have to make about the user and the electronic diary in order to create a reasonable usability specification?

Answer:

We will consider the interaction principle of guessability, which concerns how easy it is for new users to perform tasks initially. The measuring concept will be how long it takes a new user, without any instruction on the new system, to enter their first appointment in the diary. A sample usability specification is given below.

Attribute: Guessability
Measuring Concept: Ease of first use of system without training
Measuring Method: Time to create first entry in diary
Now Level: 30 seconds on paper-based system
Worst Case: 1 minute
Planned Level: 45 seconds
Best Case: 30 seconds (equivalent to now)

The values in this usability specification might seem a little surprising at first, since we are saying that the best case is only equivalent to the currently achievable level now. The point in this example is that the new system is replacing a very familiar paper and pencil system which requires very little training. The objective of this system is not so much to improve guessability but to preserve it. In the chapter, we discussed that the worst case level should not usually be worse than the now level, but we are hoping for this product to improve overall functionality of the system. The user will be able to do more things with the electronic diary than they could with the conventional system. As a result, we worry less about improving its guessability. Perhaps we could have been more ambitious in setting the best case value by considering the potential for voice input or other exotic input techniques which would make entry faster than writing.

This time within the flexibility category, we want to support the task migratability of the system. A frequent sort of task for a diary is to schedule weekly meetings. The

conventional system would require the user to make an explicit entry for the meeting each week—the task of the scheduling is the responsibility of the user. In the new system, we want to allow the user to push the responsibility of scheduling over to the system, so that the user need only indicate the desire to have a meeting scheduled for a certain time each week and the system will take care of entering the meeting at all of the appropriate times. The task of scheduling has thus migrated over to the system. The usability specification for this example follows.

Attribute: Task migratability
Measuring Concept: Scheduling a weekly meeting
Measuring Method: Time it takes to enter a weekly meeting appointment
Now Level: (time to schedule one appointment) (number of weeks)
Worst Case: time to schedule two appointments
Planned Level: 1.5 (time to schedule one appointment)
Best Case: time to schedule one appointment

In this specification, we have indicated that the now level is equivalent to the time it takes to schedule each appointment separately. The worst planned and best-case levels are all targeted at some proportion of the time it takes to schedule just a single appointment—a dramatic improvement. The difference between the worst, planned and best-case levels is the amount of overhead it will take to indicated that a single appointment is to be considered an example to repeated at the weekly level.

What are the assumptions we must make to arrive at such a usability specification? One of the problems with usability specifications, as we have stated in the chapter, is that they sometimes require quite specific information about the design to be expressed. For example, had we set one of our measuring methods to count keystrokes or mouse clicks, we would have had to start making assumptions about the method of interaction that the system would allow. Had we tried to set a usability specification concerning the browsing of the diary, we would have had to start making assumptions about the layout of the calendar (monthly, weekly, daily) to make our estimates specific enough to measure. In the examples we have provided above, we have tried to stay as abstract as possible, so that the usability specifications could be of use as early in the design life cycle as possible. A consequence of this abstractness, particularly evident in the second example, is that we run the risk in the usability specification of setting goals that may be completely unrealistic, though well-intentioned. If the usability specification were to be used as a contract with the customer, such speculation could spell real trouble for the designer.

2. Can you think of any instances in which the ‘noun–verb’ guideline for operations, as suggested in the Apple human interface guidelines for the Desktop Interface, would be violated? Suggest other abstract guidelines or principles besides consistency which support your example. (Hint: Think about moving files around on the Desktop.)

Answer:

According to the noun-verb rule, all operations that the user will execute are made up of an action (the verb) operating on a single parameter (the noun). The action (move or copy) in the case of moving a file (or copying for that matter) requires more than one argument. The move operation needs the user to first pick the icon for the file to be moved, then implicitly indicate the move operation by dragging the selected icon to the destination folder. The file to be moved and the destination folder are the nouns in this conversation. The move operation is the verb. The order noun-verb-noun is the most natural way to convey something. To strictly adhere to the noun-verb rule, we would have to specify both the target file and the destination folder prior to indicating the transfer action. That would be compatible with most other commands on the desktop in terms of input expression. Some principles of direct manipulation and familiarity, on the other hand, are more crucial. Dragging files around the desktop is akin to picking up any thing in the physical world and moving it to a new spot. Furthermore, the dragging procedure is incremental and easily recoverable; transferring to a different location can be undone inside the same operation, as the dragging can continue until the file is released. Because the verb is "move to target folder," some would argue that there is no violation of the noun-verb guideline (thus, moving is still consistent with respect to input phrase). However, a command to search a file system for files matching some criteria might be a better illustration. The action is to perform a qualified search, and the argument or noun is the set of folders or volumes on the system to be searched. This type of operation is usually described by a dialogue box that allows the user to choose the operation's characteristics (search parameters) and the folders or volumes to search in any order. The user then specifies that it is OK for the system to undertake the function after this unordered conversation is completed. This type of form-filling dialogue follows neither the noun-verb nor the verb-noun guidelines; the user's sequence is more variable than consistent.

3. Can you think of any instances in which the user control guideline suggested by Apple is not followed? (Hint: Think about the use of dialogue boxes.)

Answer:

"All activities are initiated and controlled by the user, not the computer," according to the user control guideline, which is obviously contradicted in the case of conversation boxes. When a system error occurs, a conversation box can be utilized to indicate it. The system only permits the user to acknowledge the error and dismiss the dialogue box once the error has been detected and displayed to the user in the dialogue box. With good cause, the system preempts user dialogue. The dialogue box's preemptive nature ensures that the user is aware that an error has occurred. Presumably, the only error that will occur in such a situation is preemption is justified since intrusive methods require the user's knowledge before proceeding. However, conversation boxes are not always used to communicate faults, and they might still inhibit the user from taking tasks that they would otherwise like to do. The user may be asked to fill in certain information to set parameters for a command in the dialogue box. If the user is unsure of what to supply, they are at a loss. The user may usually discover the information by navigating through another section of the system, but to do so, they must quit the dialogue box (and lose any settings they may have already entered), retrieve the missing information, and start over. This type of foresight is undesirable. This is most likely the type of preemption that this recommendation is intended to prevent, but it isn't always followed.

4. Find a book on guidelines. List the guidelines that are provided and classify them in terms of the activity in the software life cycle to which they would most likely apply.

Answer:

Mayhew's book Principles and Guidelines in Software User Interface Design serves as a source of guidelines. All criteria, in general, impose limits on the design process and should be understood throughout the requirements phase. We'll focus on which other stages (architectural design, detailed design, coding and unit testing, integration, and testing) will be most affected by the guideline in the following list. The page reference for the specified guideline is indicated by the numbers in parentheses.

- Architectural design
  – Present functionality through a familiar metaphor. (97)
  – Provide similar execution style of analogous operations in different applications. (97)
  – Organize the functionality of a system to support common user tasks. (442)
  – Make invisible parts and processes visible to the user. (95)

- ❖ Detailed design
  – Consistent dialogue style for different functions. (97)
  – Match menu structure to task structure. (144)
  – Create logical, distinctive, and mutually exclusive semantic categories with clear meanings. (150)
  – Design and organize a fill-in form to support the task. (184)
  – Consider voice synthesis as an output device when the user's eyes are busy, when mobility is required, or when the user has no access to a workstation or screen. (427)
- ❖ Coding and unit testing
  – On full-screen text menus, present menu choice lists vertically. (148)
  – In a fill-in form, use white space to create a balance and symmetry and lead the eye in the appropriate direction. (186)
  – Avoid frequent use of shift or control keys. (256)
  – Place high-use function keys within easy reach of the home row on the keyboard. (281)
- ❖ Integration and testing
  – Allow full command names and emphasize them in training, even if abbreviations are allowed. (261).

5. What is the distinction between a process-oriented and a structure-oriented design rationale technique? Would you classify psychological design rationale as process- or structure-oriented? Why?

Answer:

The difference between a process-oriented and a structure-oriented design reasoning is the information that the design rationale tries to capture. The goal of process-oriented design rationale is to capture a historically accurate description of a design team making a conclusion on a specific design challenge. Process-oriented design rationale, in this sense, becomes a separate activity from the rest of the design process. The preservation of the design's historical evolution is less important to structure-oriented design rationale. Rather, it is more concerned with giving the design activity's conclusions so that it can be done after the fact in a post-hoc and reflective manner. The task-artifact relationship is supported by psychological design reasoning cycle. The systems on which the user community performs the tasks change the tasks that they perform. A psychological design rational starts with the system's designers recording what tasks they believe the system should assist, and then developing the

system to support those tasks. The designers propose scenarios for the tasks that will be utilized to monitor new system users. The knowledge needed for the real design reasoning of that version of the system comes from user observations. The design's assumptions about the main tasks are then compared to actual use to justify or recommend changes.

The justification for psychological design is primarily a process-oriented approach. A claims analysis is all about capturing what the system's designers thought about it at one point in time and how those assumptions compared to actual use. As a result, understanding the psychological design rationale's history is critical. Designers must conduct claims analysis within the actual design process, rather than as a post-hoc reconstruction, due to the discipline necessary in completing a psychological design reason.

6. Do a keystroke level analysis for opening an application in a visual desktop interface using a mouse as the pointing device, comparing at least two different methods for performing the task. Repeat the exercise using a trackball. Discuss how the analysis would differ for various positions of the trackball relative to the keyboard and for other pointing devices.

Answer:

We examine three distinct approaches for launching an application on a visual desktop at the keystroke level. For a traditional one-button mouse, a trackball mounted away from the keyboard, and one mounted near to the keyboard, these solutions are examined. The fundamental difference between the two trackballs is that the latter does not require explicit hand repositioning, i.e., there is no time spent homing the hands between the pointing device and the keyboard.

Method 1: Double clicking on application icon.

Steps	Operator	Mouse	Trackball_1	Trackball_2
1. move hand to mouse	H[mouse]	0.400	0.400	0.000
2. mouse to icon	P[to icon]	0.664	1.113	1.113
3. double click	2B[click]	0.400	0.400	0.400
4. return to keyboard	H[kbd]	0.400	0.400	0.000
Total times		1.864	2.313	1.513

Method 2: Using an accelerator key

Steps	Operator	Mouse	Trackball$_1$	Trackball$_2$
1. move hand to mouse	H[mouse]	0.400	0.400	0.000
2. mouse to icon	P[to icon]	0.664	1.113	1.113
3. click to select	B[click]	0.200	0.200	0.200
4. pause	M	1.350	1.350	1.350
5. return to keyboard	H[kbd]	0.400	0.400	0.000
6. press accelerator	K	0.200	0.200	0.200
Total times		3.214	3.663	2.763

Method 3: Using a menu

Steps	Operator	Mouse	Trackball$_1$	Trackball$_2$
1. move hand to mouse	H[mouse]	0.400	0.400	0.000
2. mouse to icon	P[to icon]	0.664	1.113	1.113
3. click to select	B[click]	0.200	0.200	0.200
4. pause	M	1.350	1.350	1.350
5. mouse to File menu	P	0.664	1.113	1.113
6. pop-up menu	B[down]	0.100	0.100	0.100
7. drag to open	P_{drag}	0.713	1.248	1.248
8. release mouse	B[up]	0.100	0.100	0.100
9. return to keyboard	H[kbd]	0.400	0.400	0.000
Total times		4.591	6.024	5.224

7. One of the assumptions underlying the programmable user model approach is that it is possible to provide an algorithm to describe the user's behaviour in interacting with a system. Taking this position to the extreme, choose some common task with a familiar interactive system (e.g. creating a column of numbers in a spreadsheet and calculating their sum, or any other task you can think of) and describe the algorithm needed by the user to accomplish this task. Write the description in pseudocode. Does this exercise suggest any improvements in the system?