Using the Keystroke-Level Model to Estimate Execution Times

David Kieras
University of Michigan

© David Kieras, 2001

Introduction

The Keystroke-Level Model (KLM), proposed by Card, Moran, & Newell (1983), predicts task execution time from a specified design and specific task scenario. Basically, you list the sequence of keystroke-level actions the user must perform to accomplish a task, and then add up the times required by the actions. It is not necessary to have an implemented or mocked-up design; the KLM requires only that the user interface be specified in enough detail to dictate the sequence of actions required to perform the tasks of interest.

The actions are termed *keystroke level* if they are at the level of actions like pressing keys, moving the mouse, pressing buttons, and so forth, as opposed to actions like "log onto system" which is much more abstract. The KLM requires that you describe how the user would do the task in terms of actions at this keystroke level. The basic actions are called *operators*, in the sense of operators in the Model Human Processor discussion. There is a standard set of operators for use in the KLM, whose execution times have been estimated from experimental data. This presentation is based on Card, Moran, & Newell(1983), but has several extensions, especially concerning the placement of mental operators.

The Approach

The Method

The following is a step-by-step description of how to apply the KLM to estimate the execution time required by a specified interface design:

1. Choose one or more representative task scenarios.
2. Have the design specified to the point that keystroke-level actions can be listed for the specific task scenarios.
3. For each task scenario, figure out the best way to do the task, or the way that you assume users will do it.
4. List the keystroke-level actions and the corresponding physical operators involved in doing the task.
5. If necessary, include operators for when the user must wait for the system to respond
6. Insert *mental operators* for when user has to stop and think.
7. Look up the standard execution time to each operator.
8. Add up the execution times for the operators.
9. The total of the operator times is the estimated time to complete the task.

Operators and Times

The following are the standard operators and estimated times for each operator.

K - Keystroke (.12 - 1.2 sec; .28 recommended for most users). This operator is pressing a key or button on the keyboard. Pressing the SHIFT or CONTROL key counts as a separate keystroke. Different experience levels have different times for the K operator:

Expert typist (90 wpm): .12 sec
Average skilled typist (55 wpm): .20 sec
Average nonsecretarial typist (40 wpm): .28 sec
Worst typist (unfamiliar with keyboard): 1.2 sec

The average nonsecretarial typist (.28 sec) is a good design point for characterizing the typical computer user; these are people familiar enough with the keyboard to use it fluently, but are not professional-grade typists.

T(n) - Type a sequence of *n* characters on a keyboard (n □ K sec). This operator is simply a shorthard for a series of K operators, and would normally be used only when the user is typing a string of characters that is a single "chunk," such as a filename.

P - Point with mouse to a target on the display (1.1 sec). This operator represents the action of moving the mouse to point the cursor to a desired place on the screen. The actual time required can be determined from Fitts' law. For typical situations, it ranges from .8 to 1.5 sec, with an average of 1.1 sec. If great accuracy is not required, or the movement distances or target sizes are not unusual, this average can be used instead of more precise times.

B - Press or release mouse button (.1 sec). This is a highly practiced, very rapid movement. Figure .1 sec for pushing the button down or letting it up.

BB - Click mouse button (.2 sec). Pushing and releasing the mouse button rapidly, as in a selection click, counts as two B operators, for a total of .2 sec.

H - Home hands to keyboard or mouse (.4 sec). Since the targets are pretty large, and the movement well practiced, moving the hand between keyboard and mouse, and vice-versa, is relatively fast.

M - Mental act of routine thinking or perception (.6 - 1.35 sec; use 1.2 sec). Of course, how long it takes to perform a mental act depends on what cognitive processes are involved, and is highly variable from situation to situation or person to person. This operator is based on the fact that when reasonably experienced users are engaged in routine operation of a computer, there are pauses in the stream of actions that are about a second long and that are associated with routine acts such as remembering a filename or finding something on the screen. The **M** operator is intended to represent this routine thinking, not complex, lengthy, problem-solving, racking the brain, or creative meditations. In a variety of routine computer usage tasks such as word processing and spreadsheet usage, these routine pauses are fairly uniform in length, justifying the simplifying assumption that all **M**s take the same amount of time, around one sec.

Based on the available results (Olson & Olson, 1990), a good overall estimate for the duration of an **M** is 1.2 sec. Choosing how many **M**s are involved, and where they appear, is the hardest part of using the KLM.

W(t) - Waiting for the system to respond (time *t* must be determined). This is the time that the user must wait on the system before he or she can proceed. Notice that it is not necessarily the same as the time required by the system, because the user may be able to overlap other activities while the system is working.

In many cases with modern computers, the waiting time is essentially negligible. In other cases, such as a database retrieval or website usage, the delay might be substantial, but still the same regardless of which interface design is under consideration. In such cases, it may not be necessary to include this operator, either because it takes near-zero time, or it has the same value in all of the alternative designs, and so affects only the absolute, not the relative, task times.

Note: Card, Moran, & Newell call this the **R** operator, for *Response Time*; here we use **W** for *Waiting Time* to make it clear that what is relevant is how long the user must wait, not how long the system takes to respond.

Example 1. Physical Operators Only

This example consists of adding a Delete command to the Macintosh Finder menu. First, we will consider just the physical operators, deferring till later the problem of choosing and placing the mental operators.

In the current design, you delete a file by dragging its icon to the trash can icon. This is relatively slow, and you have to be able to see and point to the trash can. On a small screen, this can be awkward, because the trash can might easily be under a window. The new design idea is to add a DELETE item to the FILE menu. This would be used by first selecting one or more files, then choosing the DELETE item with the mouse. The selected files are then moved to the trash can.

The question is: *How much time will the new design save?* First we will look just at the physical operator time, and compare the current and new design in two scenarios. The first scenario is just a simple file deletion, while the second one represents the case in which the trash can is obscured.

In all of these examples, we will assume that the waiting time is negligible, and so we do not deal with any **W** operators.

Scenario 1

Assumptions. Let's spell out exactly what is involved in the first scenario, which is just the simple case of deleting a file.

> One file is to be deleted
> File icon is visible and can be pointed to
> Trash can icon is visible and can be pointed to
> Cursor must end up in the original window that the file icon was in
> Hand starts and ends on mouse
> User is average non-secretary typist (40 wpm)

Current design. The best (in fact only) procedure is to drag the file icon to the trash can icon. We need to spell out the exact sequence of actions, then we will translate this to the sequence of operators, and then determine the total time to execute this task.

> **Action sequence**
> 1. point to file icon
> 2. press and hold mouse button
> 3. drag file icon to trash can icon
> 4. release mouse button
> 5. point to original window

Operator sequence

```
1. point to file icon P
2. press and hold mouse button B
3. drag file icon to trash can icon  P
4. release mouse button B
5. point to original window P
```

Total time = 3**P** + 2**B** = 3*1.1 + 2*.1 = 3.5 sec

So we estimate that carrying out this task on the current design will take about 3.5 secs for the physical actions alone.

New design. In the new design, the procedure we intend people to use is to first select the file to be deleted, and then select DELETE on the FILE menu. We will list the actions and operators, and calculate the time as with the current design.

Action sequence

```
1. point to file icon
2. click mouse button
3. point to file menu
4. press and hold mouse button
5. point to DELETE item
6. release mouse button
7. point to original window
```

Operator sequence

```
1. point to file icon P
2. click mouse button BB
3. point to file menu P
4. press and hold mouse button B
5. point to DELETE item P
6. release mouse button B
7. point to original window P
```

Total time = 4**P** + 4**B** = 4*1.1 + 4*.1 = 4.8 sec

The new design takes about 4.8 seconds to perform the same task.

Conclusion. In this scenario, the new design is actually slower by more than a second! Some improvement!

It is easy to see that the problem with the new design is that it requires an additional mouse movement, which is pretty slow. The obvious rejoinder is to define a "Power Key" on the menu, which would allow the mouse move to be bypassed. Let's see what this variation requires for execution time in the same scenario.

Power key variation on new design. We will add a "power key" for the delete file operation, such as command-T (T for "trash"). For the sake of illustration, we assume that user uses both hands to type the power key, and so will have to move the hand from the mouse to the keyboard. The general procedure is now to select the file to be deleted, then hit the command keystroke for delete.

For the rest of the examples, we will skip the separate action sequence and directly list the actions and corresponding operators.

Operator sequence

```
1. point to file icon P
2. click mouse button BB
3. move hand to keyboard H
4. hit command key command-T KK
5. move hand back to mouse H
```

Total time = **P** + 2**B** + 2**H** + 2**K** = 1.1 + .2 + .8 + .56 = 2.66 sec

Conclusion. In Scenario No. 1, using the power key produces a noticeably faster execution time over the original design, a savings of about a second. It would be faster still if we didn't have to move between the mouse and the keyboard, which a good choice of the command keystroke would allow.

Scenario 2

What if the trash can is hidden, and has to be uncovered before the icon can be dragged to it?

Assumptions. The trash can is hidden, but we need to spell out the scenario exactly.

One file is to be deleted.
File icon is visible and can be pointed to in window A.
Trash can icon is covered by one window, B, that must be moved out of the way to see the trash can, but B must remain open on the screen.
There is room for both window A and window B on the screen at the same time.
Cursor must end up in window A
Hand starts and ends on mouse

Current design. The best procedure is to move window B out of the way, then drag the file icon to the trash can, and then return the cursor to the original window, A.

Operator sequence

```
1. point to the title bar of window B P
2. hold down the mouse button B
3. drag the window to another place P
4. release the mouse button B
5. point to file icon P
6. press and hold mouse button B
7. drag file icon to trash can icon  P
8. release mouse button B
9. point to original window P
```

Total time = 5**P** + 4**B** = 5*1.1 + 4*.1 = 5.9 sec

New design. The general procedure is the same as in Scenario 1. First, select the file to be deleted, then select DELETE on the FILE menu. Note that window B is not moved, and the cursor stays in the proper window. We can use the calculation from Scenario 1, both the menu version and the power key variation of the new design.

Total time, menu version = 4.8 sec
Total time, power key version = 2.66 sec

Conclusion. In summary, for Scenario 2:

Current design: Time = 5.9 sec
New design: Time = 4.8 sec or 2.66 sec.

If the trash can icon is hidden, the new design is much faster, by at least a second, and even more if a power key is used and hand movement is not required.

Overall Conclusions for Example 1

Clearly, the visibility of the trash can is critical to this design problem. The new design is an improvement only if the trash can is hidden or a power key is used. Adding a delete command will probably complicate the Finder interface somewhat; there is only one other file manipulation command, DUPLICATE, and this would be the only file movement command. So adding the DELETE command is a relatively big change to the Finder interface. Another ramification: do we need to implement the UNDO command for file deletion to maintain consistency?

So, we should implement this DELETE command only if:

- A hidden trash can is a frequent problem, or
- Deletions are frequent enough to justify committing a power key.

The KLM can't answer these questions, but it does tell us what we need to find out to make the decision, and gives us a useful quantification of the potential benefit of the new command to the user.

Including Mental Operators

Example 1 did not attempt to represent where users might have to think about what they are doing; it only concerned the physical operators. However, mental operations can be very time consuming, and so an interface that required less "think time" might be superior in speed, even if some extra physical actions were involved.

Including mental operators in the KLM requires identifying where the user will have to stop and think, or engage in some form of mental activity. Represent these occasions as a mental action, **M**. Doing this is tricky, and takes a lot of judgment; you have to hypothesize about how the user *thinks* about the task, rather than just what movements they must make. Following are some suggestions for how to make these decisions.

General Suggestions for Working with Ms

Consistency is key. Consistency in **M** placement is more important than absolute accuracy - be sure that you follow the same rules or philosophy in assigning **M**s to each alternative design.

Number of Ms is more important than placement. Sometimes it's pretty clear what pieces of thinking the user might have to do, but the exact times or sequence in which they do these piece might be very unclear. Note that for estimating the task time, the number of **M**s is important, not where in the sequence they appear. So explore alternative ways of how users might think about the task; if there is no difference in the number of **M**s, you can be more confident of your results.

If this doesn't help, see if the design issue can be resolved without resolving questions about **M** placement. For example, if the physical times are very different, by more than a couple of **M**s worth, you might be able to decide the design question without resolving the **M** issues.

Apples & oranges. If two designs involve very different kinds of thinking, proceed with caution. An example is having to position tab stops to create a table of numeric data in WYSIWYG vs. command-language document generators. The WYSIWYG system involves trial & error visual judgments, while the command-language system involves some kind of arithmetic reasoning. It's pushing your luck to assume that the total thinking time will be equal in these two systems, and it is real risky trying to figure out how many **M**s are involved in a consistent way for two such different environments.

The yellow pad heuristic. If the alternative designs put you into an apples & oranges situation, consider removing the mental activities from your action sequence and assume that the user has the results of such activities easily available, as if they were written on a yellow pad in front of them. Using the above tab stop example, the yellow pad would contain the tab stop positions, "already figured out." So all the user has to do is get those positions specified to the system.

This approach separates activities determined by the specific action sequences required by the interface from activities determined by what information the user must come up with. You can then try to make a decision based on considering these two issues separately. For example, you could collect some data to see how long it takes users to think up numerical tab stop settings in the command-language document generator.

Specific Suggestions for Activities that take an M

Initiating a task. The user has to pause and make a definite decision about what the task is, and what should be done. Thus users often pause before emitting a sequence of actions; this pause should be routinely represented by an **M**.

Making a strategy decision. If there is more than one way to proceed, and the decision is not obvious or well practiced, but is important, the user has to stop and think.

Retrieving a chunk from memory. A chunk is a familiar unit such as a file name, command name or abbreviation. For example, if the user wants to list the contents of directory `foo`, they need to retrieve two chunks, `dir` and `foo`, each of which takes an **M**.

Finding something on the screen. The user must pause and scan the screen for an item that they do not already know or whose location on the screen they do not already know from practice.

Thinking of a task parameter. The user must either remember or otherwise access a task parameter value.

Verifying that a specification or action is correct. Before users signal the system to proceed, they often pause and check their entry. For example, users often stop and examine a command before hitting return, or check a dialog box before clicking on the OK, or that a destination folder is reverse-video before releasing the mouse button.

Some useful conventions. Models often contain certain sequences of actions which should be represented in a consistent way. Some suggestions:

• The values for task parameters have to be explicitly obtained in a step.

• Pointing to an object on the screen should be preceded by a mental operator to locate the object.

• If something on the screen changes in response to user input, there should be a step to veryify that the desired result appeared.

Suggestions for Assigning Ms for New vs. Experienced Users

The major source of confusion in assigning Ms seems to be whether the user just zips along, or has to painfully think about everything. Clearly it depends on how experienced the user is. New users would work slowly and carefully, deliberately making decisions and struggling to remember each thing that must be recalled. Experienced users would make only the minimum required number of decisions, and would spend little time in memory retrieval or screen searches -- they have everything at their mental fingertips and know where everything is. There is clearly a big difference in the number of **M**s involved.

How do you deal with this situation? The basic advice is to consider ***both*** new and experienced users unless you are confident that you are interested in only one level of user experience. Calculate a time estimate for both levels of experience, and see if the design decision comes out the same. If not, the KLM isn't going to help you much. Here are some specific suggestions for the difference between new and experienced users -- these are exceptions to the above suggestions for activities that take an **M**.

New users verify every step. New users will stop and check feedback from system at every step, taking an **M** to do so. Experienced users skip the verification step, sometimes making errors as a result. There is some empirical evidence for this difference.

New users have small chunks, experienced users have big chunks. New users have to think about assembling commands from the basic pieces; they've not yet learned the larger patterns. Experienced users may have much more elaborate chunks, in which a whole complex command is zipped out as a single stream. For example, with the old MTS e-mail, I always used the editor to neaten up my message with `justify /file left 1 65`. This command was always the same, and I did it a lot, so it became for me a single chunk, typed in as a unit.

Remember that the only things that can be chunked together this way are things that have appeared in a constant sequence or pattern - you can't automate or group together things that happen in different ways each time. For example, suppose you justify different parts of your files, using justification commands like `justify 23 84 left 1 65` in which `23 84` specifies the specific range of lines to be operated on. Since this is different every time, you won't be able to form a single chunk for the entire command.

Also remember that the only things that could be chunked together are constant patterns that are used *very frequently* -- sequences that the user would rarely execute are unlikely to get chunked. Example: On the Macintosh, throwing a file in the trash is a candidate for a single chunk; switching off Appletalk is probably not.

Experienced users can overlap Ms with physical operators. An apparent characteristic of highly practiced performance is the ability to do more than one thing at a time if it is physically possible. For example, a practiced user might be able to visually locate an icon on the screen while homing the hand to the mouse and starting the mouse movement. In this case, you can want to drop the **M** for finding the icon visually, leaving just the physical **H** and **P** operators.

Example 2. Including Ms

To illustrate how to include Ms, we will analyze the same design problem as before, but for brevity, only portions of it. We will consider new vs. experienced users in Scenario 1, current design only. Scenario 2 will be analyzed only for an experienced user, with a comparison made between the current design and the power-key variation of the new design.

Current design - Scenario 1, New User

Assumptions. Our basic assumption is that the new user will stop and check feedback from system at every step. The general procedure such a user will follow is to find the file icon, make sure it is selected, and drag it to the trash can, making sure the trash can has been hit (reverse videos), and then verify the whole process by checking that the trash can is bulging.

Operator sequence

```
1. Initiate the deletion (decide to do the task) M
2. find the file icon M
3. point to file icon P
4. press and hold mouse button B
5. verify that the icon is reverse-video M
6. find the trash can icon  M
7. drag file icon to trash can icon  P
8. verify that the trash can icon is reverse-video M
9. release mouse button B
10. verify that the trash can icon is bulging M
11. find the original window M
12. point to original window P
```

Total time = 3**P** + 2**B** + 7**M** = 11.9 sec

If this seems long, maybe it is because you are more experienced!

Current design - Scenario 1, Experienced User

Assumptions. We will assume the following things about the experienced user. As a result, the general procedure is simply to find the file icon to be deleted and drag it to the trash can.

Experienced user thinks of selecting and dragging an item as a single operation - a chunk
The user must find the to-be-deleted icon since it is different every time
Moving icons to the trash can is highly practiced:
The trash can does not have to be located, so finding it is overlapped with pointing to it
Verifying that the trash can has been hit is overlapped with pointing to it
Final result (bulging can) is not checked since it is redundant with verifying that the can has been hit
Pointing to the original window is overlapped with finding it

Operator sequence

```
1. initiate the deletion M
2. find the file icon M
3. point to file icon P
4. press and hold mouse button B
5. drag file icon to trash can icon  P
6. release mouse button B
7. point to original window P
```

Total time = 3**P** + 2**B** + 2**M** = 5.9 sec

This seems to be in the right ballpark for an experienced user.

Conclusion. The current design can be slow for new users in Scenario 1, because there are a lot of things to find and verify. Since these are pretty consistent over time, an experienced user can be much faster.

Scenario 2, Experienced User

We will compare an experienced user in the two extreme cases of the current design and the power key variation of the new design.

Assumptions. The assumptions about the expert listed above still apply, with some additional ones.

Noticing that the trash is covered, and that it needs to be uncovered, requires thought
Finding a place to move a window requires thought
This command-key combination does not require retrieveal, since it is frequently practiced
Moving windows is also highly practiced

Current design - Scenario 2

Operator sequence

1. initiate the deletion **M**
2. notice that trash is covered and decide to uncover it **M**
3. choose a suitable empty place on the screen **M**
4. initiate the window move **M**
5. point to the title bar of window B **P**
6. hold down the mouse button **B**
7. drag the window to the other place **P**
8. release the mouse button **B**
9. find the icon for the to-be-deleted file **M**
10. point to file icon **P**
11. press and hold mouse button **B**
12. drag file icon to trash can icon **P**
13. release mouse button **B**
14. point to original window **P**

Total time = 5**P** + 4**B** + 5**M** = 11.9 sec

Thinking requires an additional 6 sec beyond the physical activities.

Power key variation on new design - Scenario 2

Operator sequence

1. initiate the deletion **M**
2. find the icon for the to-be-deleted file **M**
3. point to file icon **P**
4. click mouse button **BB**
5. move hand to keyboard **H**
6. hit command key command-T **KK**
7. move hand back to mouse **H**

Total time = P + 2B + 2H + 2K + 2M = 5.06 sec

An additional 2.4 seconds above physical time is required

Conclusion. In Scenario #2, taking mental time into account, the power key variation on the new design is much faster that the current design, taking about half the time. Quite a bit of the savings is a result of not having to think about whether and how to uncover the trash.

Implications of Operator Times for Interface Style Choices

The speed of an interface involves tradeoffs, which can be captured in terms of the operator times:

Keystrokes are fast
Ms, such as memory retrievals and visual searches, are very slow
Mouse moves are very slow
Switching hand between mouse and keyboard is moderately slow

This makes it clear that there is no single answer for which style of interface is the fastest overall. That is, command keys are fast, but only if they can be remembered quickly; that is, are easy to learn, so that eventually they will not cost an **M** to retrieve. But you quickly run out of mnemonics! On the other hand, mouse-based menus do not require memory retrieval, but do require mouse moves, moving the hand between keyboard and mouse, and visual search. All of these are relatively slow. If the user can not learn where menu items are, visual search will always be required.

But on the whole, the user will have to pay **M**s, either to retrieve commands and their arguments, or to find things on the screen, and will either make fast keystrokes or slow hand/mouse moves. The quality of an interface with regard to speed is thus whether these have been traded off in a good balance, and whether the users can in fact become faster as they gain experience.

References

Card, S. K., Moran, T. P., & Newell, A. (1983). *The psychology of human-computer interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Olson, J. R., & Olson, G. M. (1990). The growth of cognitive modeling in human-computer interaction since GOMS. *Human-Computer Interaction*, **5**, 221-265.