Relating Theory to Actual Results in Computer Science and Human-Computer Interaction

Craig S. Miller
School of CTI
DePaul University
Chicago, IL 60604
USA
Phone: +1 312 362 5085
Email: cmiller@cs.depaul.edu

To appear in Computer Science Education
Compact version — do not quote

March 14, 2003

Running Head: Relating theory to actual results

Abstract

Computer science educators are increasingly adding components that compare theoretical predictions to empirical results. If we are interested in better integrating Human-Computer Interaction (HCI) concepts into a computer science curriculum, we might look at HCI lessons that draw upon the same set of practices. With this aim in mind, I present a lesson that uses Card, Moran and Newell's Keystroke Level Model and discuss the benefit of asking students to compare its theoretical predictions with empirical results from informal usability tests.

Introduction

Despite an increasing recognition that usability is a serious problem, the study of Human-Computer Interaction (HCI) remains at the periphery of most computer science programs. If we look at what HCI entails, we see other reasons for its lack of acceptance in computer science programs. Quite simply, its study and practice require skills and knowledge that are different from those that are traditionally developed in a computer science program. Traditional computer science has its roots in mathematical problem-solving, where the problems are well-specified, the axioms accepted, and the solutions verifiable. In contrast, HCI problems are ill-defined, heuristics serve in place of axioms, and evaluating proposed solutions is a study in itself. Indeed, practicing HCI chiefly involves defining the problems, constructing the heuristics and deciding just how to evaluate the solutions.

The difference in approaches between HCI and traditional computer science can present a challenge to instructors, especially those who are trained in one approach but not the other. Moreover, instructors who champion the traditional understandings of computer science, often question whether HCI belongs in a computer science program. Even if we do not accept the traditional boundaries of these disciplines, we must nevertheless acknowledge that they are still entrenched in our curriculum and understand why there may be much resistance towards incorporating HCI elements into a computer science program.

But computer science as a discipline *is* changing. Educators are increasingly recognizing computer science as an empirical science, one where data collection and analysis play roles at least as important as formal proofs (Tichy, 1998; Reed, Miller, & Braught, 2000; Reed, Baldwin, Clancy, Downey, & Hansen, 2002). One only needs to look at any trade magazine to see that benchmarks and comparison studies are the common practice for making product claims. To make sense of these claims, computing professionals need to understand the benefits and pitfalls of empirical studies.

In contrast to traditional computer science, HCI is sometimes dismissed as lacking theoretical underpinnings, or at least the kind of theory that traditional computer scientists feel comfortable with. For example, HCI textbooks often emphasize iterative design, which often comes across as simply design by trial-and-error. If we are interested in better integrating HCI concepts with a computer science curriculum, we might look at HCI practices rooted in theory. At the same time, if we address empirical elements in computer science, we find skills and knowledge in common to both disciplines. These skills relate theory to empirical results and include the following:

- Explaining how theory applies
- Identifying theoretical assumptions
- Noting when the assumptions are violated (and to what extent)
- Devising and running realistic experiments
- Accounting for experimental error

Increasingly computer science educators are promoting lessons and assignments that involve these practices. For example, Downey (1999) reports a series of assignments where students experiment with operating system components and collect results to learn how they work. In one assignment, students run test code that uses varying array sizes to see how its execution time interacts with the cache size. Students are required to explain how the cache size affects their results. Similarly, other educators have promoted assignments where students collect empirical results and

compare it to theoretical predictions (Braught & Reed, 2002).

One avenue for better integrating HCI concepts into a computer science curriculum is to design HCI lessons and assignments that apply theory and compare its predictions to collected results. In this way, HCI instruction leverages upon practices used in teaching computer science concepts. In this paper, I present how we can use Card, Moran and Newell's (1983) Keystroke Level Model (KLM) as an example HCI lesson that relates theory to actual use. While the KLM has some real-world applications, I emphasize its pedagogical value in this article. As we will see, an exercise using the KLM involves the practices described above and has the same components that have been promoted for teaching empirical investigative skills in computer science. Having used the KLM multiple times in HCI courses, I report my experience and what other instructors might expect if they were to use this model for one of their courses. I argue that lessons that apply a theory, derive detailed predictions from it and test the predictions with actual usage have pedagogical advantages, particularly if students are expected to explain actual results in terms of the theory and account for any discrepancies. The goal is not so much to replace HCI courses but to suggest HCI lessons and assignments that could fit alongside other assignments in a variety of courses including introductory, interface development, data analysis, statistics as well as HCI courses.

Experimentation in HCI courses has been advocated before (Clarke, 1998), but here I emphasize the importance of applying theories and comparing their precise predictions to empirical results. While common in most scientific disciplines, this practice is rare in HCI courses. While the Keystroke Level Model is perhaps the best known method that is precise enough to compare to actual results, its coverage in HCI texts receives minimal attention. Shneiderman (1998) cites the KLM and mentions its usage but does not explain how to apply it. Preece *et al.* (1994) present the operators, their times and a simple example but omit the rules for placing Mental operators. Only Dix *et al.* (1998) present enough detail for a student to successfully apply the model. In none of these cases is there a discussion of how the model's predictions might compare to actual results or how lessons from the analysis might generalize to non-expert usage. Moreover, it is not used as a pedagogical vehicle for discussing issues relevant to both HCI and computer science.

Using the Keystroke Level Model

The Keystroke Level Model (KLM) produces time predictions for how long it would take an expert user to complete a particular task on a particular interface (Card, Moran, & Newell, 1983). Here an expert user is defined as someone who already knows how to use the interface and is unlikely to make mistakes completing the task. Prediction times are based on the execution times of the operators needed to complete the task. While the KLM makes absolute prediction times, it is particularly useful for comparing the predicted performances of competing designs. The KLM procedure and some examples are presented in Card, Moran and Newell (1983) and in Raskin (2000). Here I describe the process and rules as I present it to my classes. The wording and rule applications differ among my examples but all should produce the same results.

Applying the KLM

In applying the KLM, a practitioner selects a representative task for the interface to be evaluated. A task may be selected because it is frequently performed or because it is a critical task that should be completed as efficiently as possible. The practitioner then lists the physical actions needed to

Table 1: Operators for the Keystroke Level Model, adapted from Card, Moran and Newell (1983) and Raskin (2000)

Constants for primitive operators	
K = 0.2 sec	Keying: the time it takes to tap a key on the keyboard or a button on the mouse (skilled typist).
P = 1.1 sec	Pointing: the time it takes to move the mouse to a position on the display.
H = 0.4 sec	Homing: the time it takes to move the hand from the keyboard to the mouse or from the mouse to the keyboard.
M = 1.35 sec	Mental preparing: the time it takes to mentally prepare for the next step.
R	Responding: the time needed for the computer to respond.

complete the task. It might be useful to group the actions into abstract steps, but ultimately the actions need to be grounded into a series of primitive actions called operators.

The three most common physical operators are key pressing (K), pointer movement (P), and switching between keyboard and mouse (H). The actions needed for interacting with most command-line and graphical user interfaces can be decomposed into these three physical operators. Card, Moran and Newell also present a drawing operator, but this is not needed for most applications. In addition to the physical actions, the practitioner lists a response operator (R) wherever the system requires time to respond.

After the complete set of K, P, H and R operators are listed, the practitioner adds mental operators (M) that represent time needed for mental deliberation. Table 1 shows a list of the operators and Table 2 lists the rules that I have presented in my classes. I have adapted the wording and added examples to fit most of the interfaces that we discuss in class.

Once the operators are listed and the rules are applied, the practitioner counts the number of each operator and multiplies each of their counts with the corresponding time cost listed in Table 1. The time costs represent average usage. In particular, the keystroke constant is that of a skilled typist. The pointing constant assumes a typical distance and target size, but Fitts's law would produce more accurate predictions for extreme distances and sizes (see MacKenzie, 1992, for examples of its application).

I usually present a simple example in class. Figure 1 presents a Web interface that converts units of one measure (e.g. meters) to units of a different measure (e.g. yards) of the same property (e.g. length). To convert 12.3 meters to yards, the user chooses **Length** in the **Property** selection box, **Meters** in the **From** box and **Yards** in the **To** box. All of these selections are accomplished with a mouse. The figure displays the state of the interface after these actions are performed. To finish the conversion, the user types **12.3** in the text field and presses the **Perform Conversion** button with the mouse.

Table 2: Steps for applying the Keystroke Level Model, adapted from Card, Moran and Newell (1983) and Raskin (2000)

1	Fully list the K, P, H and R operators needed to complete the task.
2	Insert an M operator before every K operator.
3	Insert an M operator before every P operator unless that P operator specifies additional information to a command. For example, inserting a column break using MS Word requires the user to select the Break item from the Insert menu and then point to and click on the Column break option in the dialog box. The operator of pointing to the Column break option does not require an M operator because it specifies what type of Break should be inserted.
4	Delete an M if it precedes an anticipated operator. For example, the keying (clicking) operator of a "point and click" process is an anticipated operator.
5	If a string of MKs form a cognitive unit (e.g. a word, number, name or command), delete all of the Ms except the first one.
6	For sequences of terminators to cognitive units (e.g. a sequence of return keys or closing multiple dialog boxes), delete all Ms except the M that starts the sequence.
7	For a terminator to a command, delete the preceding M.
8	Delete an M that overlaps with an R. However, do not delete the M if the next operator depends on the outcome from R.

Table 3 shows the result of a keystroke level analysis for the above task. It assumes that none of the selections are already showing and that the user's hands are at the keyboard at the beginning of the task. The table breaks the sequence of operators into groups of abstract commands, which helps clarify the rules for placing mental operators. By grouping operators that are anticipated together (Step 4 of Table 2) or are part of the same cognitive unit (Step 5), one need only place mental operators at the beginning of each abstract command. The last abstract command (i.e. pressing the button) is considered as a terminator to entering the number of units and its mental operator has been deleted (Step 7).

Student experiences with the keystroke model

When demonstrating an example or assigning an application for analysis, I select applications whose principal tasks take 30 to 60 seconds. Often these are Web applications. Examples include an apartment-finding Web site, a currency conversion application, a course enrollment application and a site for booking airline tickets. When students perform the analysis themselves, I allow them to choose from a list of possible applications but ask them to define the actual task.

For many students, applying the KLM is their first experience at rigorously decomposing user

Table 3: Example Keystroke Level Analysis for the Unit Converter

	Select length in Property selection box
M	Prepare for property selection
H	Move hand from keyboard to mouse
P	Move mouse to selection tab
K	Click on selection tab
P	Move mouse to Length
K	Click on Length

	Select meters in From selection box
M	Prepare for selecting meters
P	Move mouse to selection tab
K	Click on selection tab
P	Move mouse to Meters
K	Click on Meters

	Select yards in To selection box
M	Prepare for selecting yards
P	Move mouse to selection tab
K	Click on selection tab
P	Move mouse to Yards
K	Click on Yards

	Key in the number of units
M	Prepare for entering units
P	Move mouse to text field
K	Click mouse to activate text field
H	Move hand from mouse to keyboard
K	Type '1'
K	Type '2'
K	Type '.'
K	Type '3'

	Press convert button
H	Move hand from keyboard to mouse
P	Move mouse to button
K	Click on button

Calculation to predict task completion time:
12 * K + 8 * P + 2 * H + 4 * M
= 12 * 0.2 + 8 * 1.1 + 2 * 0.4 + 4 * 1.35
= 17.4 sec

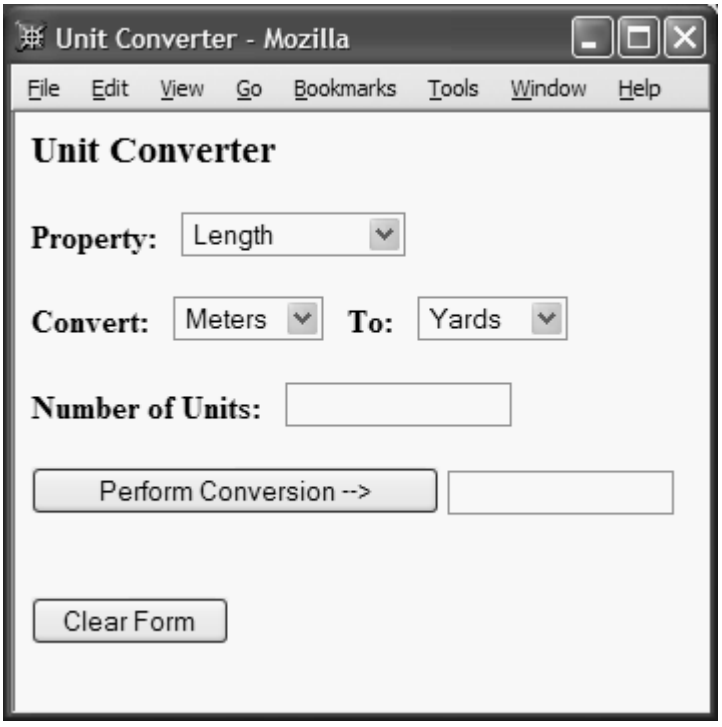

Figure 1: Simple interface for converting units

commands. They are often surprised at the number of small steps that are needed to complete even the most basic GUI tasks. Yet, after an initial exposure, students experience little difficulty in fully listing the physical operators.

Difficulty and variation of interpretation come with placing and removing the mental operators. Students need to carefully consider how commands are segmented, what constitutes a "cognitive unit", and whether a keystroke is a terminator. Grouping the operators into sets of abstract commands is useful and presents some guidance for applying the rules, but, as Card, Moran and Newell note, these decisions require some judgment from the practitioner and may depend on the user. Here some discussion is useful and presents an opportunity to apply some concepts from cognitive psychology. Discussing the size of the cognitive unit motivates teaching memory chunks and how their size increases with learning (Simon, 1974).

I typically assign a KLM problem to students working in groups of three. This arrangement allows students to discuss the rules and often requires them to explain to others why they think a certain rule applies. Generally a group of three can finish a KLM analysis on a 30 second task in a half hour and almost always within an hour.

Comparing theory to actual results

Once students have added up the times of the operators and have arrived at a prediction, they are asked to collect actual times of users performing the tasks. While not necessary, I ask student groups to "contract out" the data collection to another group. In this way, each group of students perform a KLM analysis and then collect timings on a different task for a different group. One

Table 4: Summary of student results in one class

Number in group	Type of Task	KLM Prediction	Confidence Interval from Timings	Number of Timings
3	Currency conversion	13.0	8.7 — 21.2	12
3	Course scheduling	29.9	12.3 — 45.7	13
3	Airline ticket purchase	31.5	17.5 — 28.5	18
4	Currency conversion	37.1	17.3 — 24.7	13
3	Currency conversion	10.7	22.4 — 32.0	13

Prediction and confidence interval times are presented in seconds.

advantage of asking a different group is that it requires the first group to successfully specify the task and its circumstances.

To collect timings, students run informal usability tests. The preparation involves writing instructions and setting up the computer for the task. Because the KLM analysis makes predictions for expert users, I advise students to incorporate some initial practice time before the participant performs the timed task. I supply stopwatches and lab sheets for collecting the timings. Students take turns administrating the tests and acting as participants for other tasks. They can easily collect 15 timings within an hour.

Using statistical software, students type in the times to calculate the average and a 95% confidence interval for the average. Even though they are required to have taken statistics course before, I review what a confidence interval means. Among other things, the confidence interval emphasizes the existence of error that comes from a sample of collected times.

Table 4 presents a set of student results from one class.[1] The KLM predictions are those prepared by students for the first time and without the benefit of knowing the empirical results. Since students were asked to specify the task, the actual tasks varied even for those of the same type. For example, a currency conversion task might require the user to specify a credit card rate instead of a cash rate. Except for one group of four, most KLM analyses were done in student groups of three. The first two groups have KLM predictions that fall well within the confidence interval calculated from the set of actual timings. The third group is just outside of the range. The fourth and fifth groups fall substantially outside of the range by respectively over-predicting and under-predicting the collected timings.

The results from this class are consistent with my experience in previous classes, where roughly half the student predictions fall within the confidence interval. While it is gratifying when the theory's predictions match the empirical findings, there is perhaps more pedagogical utility in asking students to account for discrepancies between the empirical result and the model's prediction. With as little prompting from me as possible, I ask students to think of reasons why the predicted time and the collected average time might differ. Here are some possibilities:

- **Because they are being timed, some participants race through the test faster than their typical pace.** Possible remedies include altering the instructions to inform the participant

[1] Thanks to Peter Wiemer-Hastings, who shared these figures from his HCI class at DePaul University. He followed the procedure I have outlined in this paper.

that the test is not a race and that he or she should perform the test at their typical pace.

- **Some participants did not receive enough practice before completing the task.** In these cases, the participants were not yet experts and probably spent more timing thinking their way through the task than users who are well practiced with the task. When discussing this possibility, it is important to remind students that the KLM is meant for predicting times of expert usage. An instructor may want to prescribe a training regiment before each participant is timed on the actual task. Card, Moran and Newell (1983) note that their participants were at ease with the task after three or four practice trials.
- **The KLM time constants do not match the skill level of the users.** For example, the keystroke rate assumes that of an average skilled typist. Some users, including some of the students, may not be skilled at typing. Note that the KLM analysis can account for users at varying levels of typing skill through the use of different time constants. However, the analysis assumes expert usage in the sense that the user knows the correct sequence of operators for accomplishing the task.
- **The rules for the M operator were not judiciously followed.** In my experience, students tend to err on the side of not removing enough M operators and produce predictions that overestimate the actual times. Some instructors may choose to restructure the mental operator rules so that they are easier to remember. For example, it might be reasonable to organize steps 3 through 7 as special examples of an automatic sequence, whose physical operators are not interrupted by an mental operators.
- **The task is poorly defined.** Sometimes the analysis group does a poor job specifying the task to the test group. The actual start of the task may not be clear. Sometimes it is not clear whether the time of the computer's response should be included.

In the case of the collected results shown in Table 4, a look at the students' KLM analysis reveals that the fourth group did not remove enough M operators because they missed some cognitive chunks. In contrast, the fifth group had removed all M operators, which may not be justified given the number of high-level steps needed to complete the task. In my experience, the actions of the fourth group is common whereas it is more unusual for students to over-apply the rules for removing M operators.

Asking students to explain discrepancies forces them to scrutinize both the model and the empirical data collection. They discover limitations of both. At first pass, it may seem that the KLM only addresses a narrow range of usability concerns. To be sure, it only applies to expert usage and only addresses one dimension of usability, that of efficiency as measured in terms of task completion time. Yet, by applying the model and comparing it to collected results, some principal ideas in HCI are addressed. For example, students learn that empirical tests critically depend on how specific the instructions are, which participants are selected, and how many of them are tested. For applying theoretical models, students learn to assess whether model assumptions are valid for the situation. Because the keystroke analysis gives students the language for explaining the consequences of these issues, they are more likely to retain the concepts (see Chi *et al.*, 1989, for a review of how explanation benefits learning).

The KLM also opens the discussion on trade-offs between interaction styles. Because the pointing operator requires 1.1 seconds compared to 0.2 seconds for the keystroke operator, keying is identified as a more efficient means for entering information to a computer. At first, this observation comes to a surprise to students who think that command-line interfaces are harder to use. With some discussion, students learn that keying may provide a more efficient means for specifying

commands, but it has a learnability cost for novice users, who have yet to learn the commands. It may be worth pointing out that many interface designers resolve the trade-off between efficiency and learnability by providing users with alternate means of selecting commands: by mouse-driven menu selection *and* by keyboard shortcuts (also called accelerators).

My students use stopwatches for collecting task completion times. Their use provides a quick and easy method for collecting times in any computer lab. Also, because the times represent a simple quantitative measure of efficiency, descriptive statistics (e.g. mean and variance) provide an easy means for summarizing a number of trials. However, an instructor may choose to have students further explore discrepancies between the KLM analysis and the collected timings. In this case, the instructor would need a method for collecting richer data. Software that captures keystrokes and their times would allow students to determine average times for each of the component actions. Screen capture software could also be used to analyze user actions as well as review user comments for those packages that record audio. Use of these methods may also motivate a discussion on the trade-offs between quantitative measures and rich qualitative data.

Discussion

The Keystroke Level Model may be the best model for applying a theory and comparing its precise predictions to actual results, but there are other theories and practices that make testable claims which could be compared to actual results. Fitts's law (MacKenzie, 1992) for mouse pointing times and Hick's law (Card et al., 1983) for menu decision times both make precise predictions that could be compared to collected times. Collecting data for these laws would require keystroke-capture software to automatically record selection times since the user event happens so quickly (on the order of 1 second).

It is also possible to look at practices that make qualitative claims instead of precise quantitative predictions. For example, card sorting is often prescribed as a means for hierarchically organizing a Web site (see Brink, Gergle & Wood, 2002, for a description of its use). Comparing the resulting structure with alternate structures in usability tests would be useful for critically evaluating the assumptions and benefits of the practice. As another example, I have students first apply inspection methods for identifying potential usability problems. Later they perform usability tests and can check whether the predicted problems actually occur in the tests. Unfortunately these practices with qualitative claims lack precision and theory. While reflecting on comparable results from independent methods has pedagogical value, the discrepancies between predictions and empirical results are not always so evident and, when they are noticed, they do not offer a rich language in which to explain the discrepancies. However, as HCI evolves as a discipline, we may be able to increasingly use new, more precise theories in our courses.

As we look to new theories, we may want to introduce them using the same methods that we are increasingly using to teach computer science concepts. That is, we ask students to apply a theory, compare the theoretical predictions to collected data, and account for any discrepancies. Both theory application and empirical data collection have their limitations. Perhaps by unifying the discussion of these methods using concepts from HCI and traditional computer science, we can produce a more integrated program of study.

References

Braught, G., & Reed, D. (2002). Disequilibration for teaching the scientific method in computer science. *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education* (pp. 106–110). New York: ACM Press.

Brink, T., Gergle, D., & Wood, S. D. (2002). *Designing Web sites that work: Usability for the Web*. New York: Morgan Kaufman.

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

Chi, M. T. H., Bassock, M., Lewis, M., Reimann, P., & Glaser, R. (1989). Self-explanation: How students study and use examples in learning to solve problems. *Cognitive Science*, *13*, 145–182.

Clarke, M. C. (1998). Teaching the empirical approach to designing human-computer interaction via an experiential group project. *Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education* (pp. 198–201). New York: ACM Press.

Dix, A. J., Finlay, J. E., Abowd, G. D., & Beale, R. (1998). *Human-computer interaction*. New York: Prentice Hall Europe, second edition.

Downey, A. B. (1999). Teaching experimental design in an operating systems class. *Proceedings of the 30th SIGCSE Technical Symposium on Computer Science Education* (pp. 316–320). New York: ACM Press.

MacKenzie, I. S. (1992). Fitts' law as a research and design tool in Human-Computer Interaction. *Human Computer Interaction*, *7*, 91–139.

Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., & Carey, T. (1994). *Human-computer interaction*. New York: Addison-Wesley.

Raskin, J. (2000). *The humane interface: New directions for designing interactive systems*. Reading, MA: Addison-Wesley.

Reed, D., Baldwin, D., Clancy, M., Downey, A., & Hansen, S. (2002). Integrating empirical methods into computer science. *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education* (pp. 48–49). New York: ACM Press.

Reed, D., Miller, C., & Braught, G. (2000). Empirical investigation throughout the cs curriculum. *Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education* (pp. 202–216). New York: ACM Press.

Shneiderman, B. (1998). *Designing the user interface: Strategies for effective human-computer interaction*. Reading, MA: Addison-Wesley, third edition.

Simon, H. A. (1974). How big is a chunk? *Science*, *183*, 482–488.

Tichy, W. F. (1998). Should computer scientists experiment more? *Computer*, *31*, 32–40.