Professional ViewPointz

Keystroke Level Modeling as a Cost Justification Tool

By

Deborah J. Mayhew, Ph.D.

Introduction

Keystroke level modeling (Card, Moran & Newell, 1983) is one of a variety of cognitive modeling techniques that have been reported in the literature over the last two decades. Cognitive modeling, simply put, involves identifying and counting all the discrete human operations – physical (e.g., mouse click, keystroke), cognitive (e.g., read or speak a syllable of text, make a mental comparison) and perceptual (e.g. locate something on screen) – that a user must execute in order to most efficiently accomplish a specific task on a specific user interface design. System response time operators are then added to the model where appropriate. Time values for each operator (available in the literature for human operators) are then plugged into a task model to predict a total task time. The total task times generated by such models predict the fastest time (on average) that highly trained and experienced users will be able to perform a given task on a given user interface with a given set of system response times, assuming they perform the task with no errors.

Like formal usability testing, modeling is a method for assessing the expected usability of a particular user interface design. Formal usability testing is an excellent technique to use during product development when the main usability goal is *ease of learning*. This is because it is relatively easy to find potential users who are untrained and inexperienced with a proposed design to participate in your study, and testing with such users allows you to discover what about your design is unintuitive and difficult to learn for users who haven't had the benefit of training, user manuals and frequent practice and experience. Public Web sites such as e-commerce sites tend to have casual and infrequent users, and are an example of a type of software application in which ease of learning for novices is a much more important usability goal than proficiency of well trained and highly practiced experts. Thus, formal usability testing is an excellent technique to use early in the development of such sites.

However, in many cases the key usability goal of a Web-enabled application (e.g., a traditional business application for use by internal users that just happens to be web-enabled, as opposed to a public Web site) – as well as of many applications built on other platforms - is not ease of learning for new or casual users, but *ease of use* – i.e., productivity – for well-trained, highly experienced, proficient and high frequency users. This aspect of usability is much harder to measure by formal usability testing early in the design process for a new application, because you never have well-trained, highly experienced and proficient users of a system that nobody has used yet (precisely because it is still in the early design stage.) It is very hard – perhaps impossible - to accurately simulate peak proficiency usage when a design is only in the prototype stage – let alone when it is only on paper.

Thus, cognitive modeling provides a useful and practical technique for assessing a proposed user interface design against ease of use goals (i.e., goals for average task time for trained, high frequency, expert users.) It can easily be applied long before implementation or even prototyping, at a point in the development process at which formal usability testing for ease of use is just not practical.

A few scenarios come to mind in which cost justifying one specific, concrete user interface design relative to another could be a very productive exercise. First, many development projects are intended to replace older applications, rather than to automate work that has not previously been automated. In this case, before building the replacement application, it would be invaluable to be able to predict with some level of confidence that a proposed application user interface design will actually result in greater user productivity levels than the existing application, to the extent that the new development effort will in fact in time pay off. Within this scenario, it is also true that if

modeling of the initial proposed design does *not* predict a level of improvement in productivity relative to the existing applications that will in fact cost justify the development and implementation of the new application, the models will also provide insights into *why* the hoped for improvement is not predicted to be realized, which in turn can drive redesign ideas for the new application. Modeling can thus be applied iteratively to evolve the proposed design until it in fact *does* achieve predicted improvements in productivity large enough to cost justify the development effort. All this can potentially be accomplished before development begins in earnest.

Second, when an organization is comparison shopping in order to purchase a commercial software package, it would be invaluable to be able to predict with confidence which of the competing packages would support the highest level of user productivity once users are trained and experienced, as this will likely result in the greatest return on investment for the purchase. Thus modeling can be used to cost justify the purchase of one competing package over others.

And third, during the design and development of a new application, inevitably competing user interface design ideas arise. Modeling can be applied when user interface design is still just on paper to predict which of any number of competing design ideas will result in the greatest user productivity, thus cost justifying one of the competitive design proposals relative to all others. Again, this can help insure a particular desired return on investment for the overall development effort by helping to optimize the design to achieve business goals for the development project.

This article provides a high level overview of a case study which illustrates the adaptation and use of the keystroke level modeling technique to cost justify the development of a new Web-enabled application intended to replace a set of old mainframe based applications. While the case study has been heavily disguised, it is based closely on an actual project, and the results reported here are in important respects very real. A much more detailed version of this case study will appear in a forthcoming second edition (Bias & Mayhew, 2004) of the book Cost Justifying Usability (Bias & Mayhew, 1994.)

Case Study

The mail processing department of a credit card company was planning to replace a collection of 14 disparate mainframe-based applications, all with different user interfaces, that mail processing clerks (“clerks”) currently used to process incoming requests and payments from account holders. The replacement was to be a single integrated web-enabled application with a consistent and improved user interface. The business case for this replacement project was premised on an expected increase in clerk productivity which over a period of time would pay for the cost of the development effort and then continue to accrue cost savings for the company.

The new application was intended to support clerks that processed incoming mail from account holders. These clerks performed a variety of tasks such as:

- Accept and process monthly payments
- Accept and process requests for balance transfers from other credit cards
- Accept and process requests to add new credit cards to an account holder’s account
- Accept and process requests to close out an account
- Accept and process requests to refuse a fraudulent charge

In all tasks, paper request forms and checks received from account holders by mail were scanned into the system by other staff members, and the images of these forms and checks appeared as tasks in a work queue on the workstations of the clerks. The clerks would bring up the images for a given task on their screen, and then open up and work in a variety of other applications to get the requests documented in the images recorded in the appropriate databases.

The business manager overseeing the replacement development project worked out a business case to justify the project cost. Based on knowledge of the current yearly volume of transactions for each mail processing task (e.g., accept monthly payments, add a new card, etc.) and on identified opportunities for increasing productivity in the proposed Web-enabled application, she set a goal for each task that would have to be met for the overall cost justification of the new application to become a reality. These goals were expressed as a required productivity gain for each task on the proposed application relative to the existing applications. For some tasks, this goal was an 18% gain, for some it was a 10% gain, and for others it was a 0% gain (that is, at least no loss.) In other words, the goals were for trained, frequent and highly practiced mail processing clerks to be able on average to perform specific tasks anywhere from 0 – 18% faster on the new Web-enabled application than they could currently do on the multiple mainframe based applications they were using. Unless these specific task goals were met, the cost of the overall development effort would not in the end be justified, at least financially.

Having set these goals, the business manager next began to wonder if and how she could be assured that these task productivity gain goals would in fact be achieved by the new application *before* building and launching it, rather than waiting until after launch and possibly finding out that the goals had not been met, and that significant additional time and cost would be required to redesign and rebuild the new application to meet goals, further eating into the return on investment. She turned to her internal usability engineering staff, who were already stretched very thin supporting this and various other internal development projects. Together the business manager and usability engineering manager decided to look for an outside consultant to help them assess whether or not the current proposed design of the new Web-enabled application would in fact result in an application that would meet the identified productivity goals. This is where I came in.

We carried out the project very collaboratively, in the following steps:

1. Model Tasks
2. Run Existing Applications Productivity Test and Use Results to Refine Modeling Technique
3. Compare Modeling Results to Goals
4. Address Unmet Goals
5. Run New Application Productivity Test

Model Tasks

For each of eight user tasks identified as high priority by the business manager, we modeled the task both on the existing and on the proposed applications. The steps in the modeling process were as follows:

1. **Identify Task Versions.** Business management decided on a particular version of each task to model. This was important because the modeling technique captures an exact sequence of user interactions and predicts an overall task time for that specific sequence. General tasks have many possible variations. For example, in the Balance Transfer task, sometimes account holders want to transfer balances from a single other credit card, while other times they want to transfer balances from multiple other credit cards. Similarly, in the address change task, sometimes an account holder has multiple addresses (e.g., primary home, vacation home and business) and sometimes they only have one. Sometimes they want to change just one while sometimes they want to change more than one. However, a given model can only capture one of each of these potential variations at each point where they occur. Thus, business management first had to decide within each general task, exactly which very specific version of it they wanted to model.
2. **Generate Task Scenarios.** Since I was working remotely and did not have access to either the existing applications or prototypes of the proposed application, business staff

then generated "task scenarios" which documented for me how the selected version of a task would play out on both applications. We worked out a documentation format that consisted of a sequence of screen shots pasted into Microsoft PowerPoint. Each slide presented a screen shot in the sequence, with notes attached to it describing generally how the user would interact with that screen at that point in the task.

3. **Prepare Draft Models.** Based on the task scenarios, I then generated two draft models for a given task – one on the existing applications, and one on the proposed application.
4. **Conduct Model Walkthroughs.** The task scenarios were rarely complete and detailed enough to allow me to generate complete and accurate models, but they were more than adequate to support the generation of draft models. After drafting models, I would get on the phone with the business experts who had generated the task scenarios for the tasks, and we would walk through my draft models together and they would clarify and correct as necessary.

I created the models in Microsoft Excel. Each task was captured in a separate Excel file. There were three worksheets in each spreadsheet file, one capturing all operator time values to be plugged into the models, one documenting the model of the task on the existing applications, and one documenting the model of the task on the proposed application.

The models included the set of operators presented in Figure 1. The time values per operator given in Figure 1 represent an initial set used in the modeling, which later was changed (see below.)

Figure 1: Model operators with initial time values

Operator	Description	Time Value (Seconds)	Source
K	Single keystroke (average skilled typist, 55 words per minute)	0.20	Card, Moran and Newell (1983)
C	Single mouse click (finger down, finger up)	0.12	Card, Moran and Newell (1983)
P	Point with mouse (hand already on mouse, move mouse to move cursor across screen to a given target)	1.10	Card, Moran and Newell (1983)
H	"Home" – move hand from one input device (e.g., keyboard) to another (e.g., mouse)	0.40	Card, Moran and Newell (1983)
M	Mental operator – only used to capture time taken to move eyes across windows	1.35	Card, Moran and Newell (1983)
W1Existing	System response time ("Wait") for: e.g., field or window to take focus in response to mouse click	0.25	Internal measures of existing applications
W2Existing	System response time ("Wait") for: e.g., display of a new page with no lookup	0.50	Internal measures of existing applications
W3Existing	System response time ("Wait") for: e.g., display of a new page with simple lookup or carrying data from previous page	1.50	Internal measures of existing applications
W4Existing	System response time ("Wait") for: e.g., display of a new page with database search	3.00	Internal measures of existing applications
SP	Speak/read a syllable in a highly practiced sentence	0.13	John (1990)
SU	Speak/read a syllable in an unpracticed sentence	0.17	John (1990)
Scan	Scan a page of text	5.00	Personal estimate

Run Existing Applications Productivity Test and Use Results to Refine Modeling Technique

Next – and actually somewhat in parallel with our modeling effort - we carried out a productivity (ease of use) test on the existing applications. Our purpose here was two-fold: eventually we planned to compare actual existing applications task times to actual proposed application task times as a final validation of the proposed application, so the data collected now on the existing system would be used in that comparison down the line.

However, even though it was to be as much as a year later before we could run a productivity test on the implemented proposed application, we ran the existing applications productivity test earlier rather than later because we planned to use the data from it to refine and validate our modeling technique, in order to increase our confidence in our comparison of models of tasks on the two applications. That is, if we at least knew that our models of the existing applications were relatively accurate, we would have more confidence that our models of the proposed application – and thus the predicted productivity gains (or losses) of the proposed relative to the existing applications – were also relatively accurate.

Once we had actual average task times for each task on the existing applications, we compared these to the modeled times and computed error rates for the models of each task. The error rate for a task was computed as the *model* time minus the *actual* time, divided by the *actual* time. Initially some – although not all – of the error rates seemed a little high – over the 20% typically reported in the literature. To address these high error rates, a variety of "what-if" scenarios were played out with the models, varying the time values assigned to the operators "K" (typing speed), "M" (mental operations) and "Scan" (time to scan a page of text.) The time values for these three operators that yielded an optimized set of existing applications model error rates across all eight tasks were "K" set to 0.12 seconds (which equals 90 words per minute - it had originally been set to 0.20 seconds, or 55 words per minute), "M" set to zero seconds (it had originally been set to 1.35 seconds), and "Scan" set 1.0 second (it had originally been set to 5.0 seconds.)

Another way of putting this is that if we assumed that users were faster typists and faster scanners than assumed in the original models, and if we assumed that in effect these users could "parallel process" any very simple mental operations with the physical ones (thus making it unnecessary to calculate in the time of any simple mental operators), then the existing applications models predicted actual times much more accurately, that is, with lower error rates, across all 8 tasks.

Thus in the end, we used the time values for operators mentioned above, rather than the original time values given in Figure 1, to generate our predictions for task times on both the existing and the proposed applications.

It is important to note that in refining our modeling strategy we only altered the time values of those operators not firmly established in the literature. The "K" operator, which represents typing speed, by definition, will vary across user populations. The "Scan" operator is one we made up, not reported in the literature at all. And for the "M" operator, we did not change the well established time value of 1.35 seconds – rather we simply did not include the operator in our models at all in the end, under the assumption that these very experienced users would always be able to "parallel" process these simple mental operators with the physical ones. For all other operators we retained time values well established in the literature.

The spreadsheet in Figure 2 shows, for each of the 8 tasks, the *modeled* total task time on the existing applications (row 4), the *actual* total task time on the existing applications (row 5), the number of test users ("N") and trials ("(t)") in the productivity test on which the actual total task time was based (8 users were run with three trials each, but data from some trials were thrown

out because users did not follow the task interaction sequence that was modeled) (row 6), and the error rate of the existing applications model (calculated as the *model* time minus the *actual* time, divided by *actual* time) (row 7.) Here it can be seen that across the 8 tasks, the error rates of the final models of the existing applications are all below 20%, and some are quite low indeed.

Figure 2: Summary of final modeling results and productivity tests across 8 tasks

	A	B	C	D	E	F	G	H	I
1		1	2	3	4	5	6	7	8
2		Accept Monthly Payment	Balance Transfer	Add Card to Account	Close Account	Change Address	Cash Advance	Refuse a Charge	Credit Line Increase
3									
4	Existing MODELED	5.97	10.33	10.23	17.58	5.61	12.35	12.11	12.37
5	Existing ACTUAL S	9.45	18.72	13.98	14.33	4.53	6.65	14.98	11.89
6	Existing ACTUAL N (t)	8 (22)	8 (19)	5 (14)	6 (18)	8 (21)	7 (21)	8 (23)	7 (18)
7	Existing ERROR RA E	-3 35%	16%	18 23%	2 31%	-16 36%	9 90%	14 90%	-18 23%
8									
9	Proposed MODELE	3.25	16.68	16.49	19.88	6.61	3.26	9.27	10.17
0	Proposed ACTUAL	5.55	19.09	10.23	19.23	5.21	6.22	9.32	13.49
11	Proposed ACTUAL N (t)	8 (23)	7 (15)	8 (20)	5 (12)	6 (18)	8 (20)	5 (13)	8 (24)
12	Proposed ERROR R TE	38 36%	-2 21%	8 21%	18 91%	3 10%	8 17%	12 54%	-18 19%
13									
14	GOAL % DIFF	-10%	-18%	-18%	0%	0%	0%	-18%	0%
15	MODELED % DIFF	13%	24%	26%	2%	2%	-9%	32%	-2%
16	ACTUAL % DIFF	23%	21%	16%	12%	17%	11%	31%	-5%

Compare Modeling Results to Goals

Having refined and validated our modeling technique by comparing modeled times to actual times on the existing applications, we next compared the final *modeled productivity gains* (or losses) of the proposed application relative to the existing applications for each task, to the *productivity gain goals* for each task.

The spreadsheet in Figure 2 presents the modeled total task times for each of the 8 tasks on both the existing (row 4) and proposed (row 9) applications, and the percent difference between them (proposed modeled minus existing modeled, divided by existing modeled - row 15.) These percent differences between models were compared to the goal percent differences for each task, also presented in the spreadsheet (row 14.) The modeled percent differences are presented as red (and positive) numbers when the proposed application was predicted to be *slower* (that is, take more time) than the existing applications, green (and negative) when the proposed application was predicted to be *faster* (that is, take less time) than the existing applications, and yellow (and positive) when the proposed application was predicted to be *slower* than the existing applications *but not by much*.

An inspection of these numbers in the spreadsheet shows that on tasks 2, 3, 6, 7 and 8, the models predicted that the productivity goals for the proposed application, given the user interface

that was modeled, would be met and in fact exceeded, in some cases dramatically. This of course was good news to the business manager. On the other hand, task 1 was predicted to fail rather dramatically to meet the productivity goal assigned to it, and tasks 4 and 5 were also predicted to fail to meet goals, although only minimally. We next focused our attention on those tasks that were predicted to fail to meet productivity goals.

Address Unmet Goals

For tasks in which the models predicted that productivity gain goals would not be met, we analyzed the models in detail to determine why and to suggest redesign directions.

The one task that stood out dramatically in the spreadsheet in Figure 2 was task 1, the Accept Monthly Payment task. The goal for this task was that users would be *10% more productive* on the proposed application relative to the existing applications, but the models predicted that given the proposed user interface design for this task, in fact users would be *13% less productive.*

We analyzed the models for this task to determine exactly what part of the proposed user interface was accountable for this predicted productivity loss. For example, we found that 4.92 seconds of the difference between the two models (which totaled 7.28 seconds) occurred because on a single page, more operators were required on the proposed application to place the cursor in fields, tab between fields and execute a Bank Number search. Other differences, similarly analyzed and documented, accounted for the remaining difference of 2.36 seconds, and these were documented in a similarly detailed fashion. The designers then revisited the design of the Accept Monthly Payment task on the proposed application and looked for ways to eliminate the identified additional operators.

This sort of analysis was also carried out for tasks 4 and 5, which were also predicted by the models to fail to meet productivity goals, and design changes were made to these task user interfaces as well.

In theory, we could have then remodeled these tasks on the redesigned proposed application to predict if the redesigns would in fact result in achieving the increased productivity goals for these tasks. Instead, however, the business manager decided to simply go ahead and make changes driven in this way by the model analyses, and then wait until it was possible to perform a productivity test on the proposed application beta version (which reflected the design changes) to determine whether in fact the goals had been met by the redesigns.

Thus, the proposed user interface for 3 of the 8 tasks (tasks 1, 4 and 5 in Figure 9) were redesigned in response to the modeling results. It is important to note that they were *not* redesigned in ways that introduced inconsistencies into the overall conceptual model and page design conventions of the proposed application as a whole. Care was taken to eliminate operators and streamline interactions without violating the overall user interface architecture of the proposed application.

Run Proposed Application Productivity Test

Some months later, after redesign and implementation of a beta version of the proposed application, we ran a productivity test to determine actual productivity on the proposed application, as a final validation of the productivity gain goals, as well as of the model predictions.

We had to conduct this productivity test slightly differently than the one on the existing applications, since there were in fact no highly trained and experienced users of the proposed application yet – we ran the test as soon as the beta version was stabilized, as always hoping to get information on productivity gains (or losses) sooner rather than later, when changes would be

cheaper and easier to make. Thus, we had to try to simulate expert usage. We did this by providing test users with training and an opportunity to practice prior to running our test. Other than that, the testing was run just as for the productivity test of the existing applications. We were not entirely sure we would be able to get users up to potential expert peak performance in a brief training session, but in fact it appeared that we were. This was good both from the point of view of simulating potential productivity, and as a sign of how easy to learn the proposed application would be.

Figure 2 shows the data from the productivity test on the proposed application. The last row (row 16) in the spreadsheet in Figure 2 shows the percent difference between the *actual* average total task times on each task on the existing and proposed applications. Note that these are real performance differences – no modeled results are involved here. Note also that for all 8 tasks, the proposed application is showing a productivity gain – in spite of the fact that the training and practice sessions were relatively brief.

Remember that task 1, address change, had showed a productivity loss of 13% when modeled (row 15, column B in Figure 2.) Also remember that this productivity loss was analyzed in detail by comparing models, and that the user interface to the proposed application was then redesigned to address this productivity loss. In the productivity test on the *redesigned* proposed application, it was revealed to show a 23% productivity *gain* relative to the existing applications (row 16, column B) – surpassing the business goal of a 10% gain (row 14, column B.) Clearly the redesign succeeded in solving the problem the modeling had revealed.

Also note that the error rate of the model for the proposed application relative to the actual productivity data on this task (row 12, column B) is quite high (38.86%.) This in part reflects the fact that the user interface modeled and the user interface tested for the proposed application were not the same – the one tested had in fact been redesigned in response to the modeling results. We did not bother, as stated earlier, to go back and remodel after redesigning (as we could have if it was still going to be a long time before a beta version was available), we simply went next to getting actual productivity data.

Tasks 4 and 5 were the other two tasks – like task 1 - that were revealed by the modeling to fall short of business goals, and that were redesigned in response to the analyses the models facilitated. It can be seen in Figure 2 (row 16, columns E and F) that again for these two tasks, while the model results showed a productivity loss for the proposed application as modeled, the productivity tests showed a significant productivity gain for the proposed application as redesigned in response to modeling results and analyses.

Thus, the productivity test on the proposed application in its beta form – even though test users were minimally trained and experienced and the beta application was not 100% stable – validated that both the original proposed application design on a number of tasks, and a redesign of other tasks, would succeed at meeting the business goals that in turn would cost justify the whole development effort.

It might also be noted that on tasks whose design did not change from the design modeled (i.e., tasks 2, 3, 6, 7 and 8), error rates for the proposed application models are reasonable, and *modeled* percent differences are fairly consistent with *actual* percent differences. That is, the refined modeling technique in the end seemed to do a pretty good job predicting actual productivity gains.

Conclusions

Modeling proved to be a valid and very useful cost justification technique on the project reported in the case study above. It allowed the business manager to predict fairly accurately - long before

the proposed application was developed and launched – whether or not productivity gains required to cost justify the new development effort would in fact be achieved. In addition, it enabled an analysis of those tasks predicted to fail to meet business goals, which in turn drove redesign to achieve those goals – again, long before launch, when it was much cheaper and easier to make design changes.

In the end, both the credit card company and I felt that a valuable tool had been refined and validated that could be used in a similar way in the future within this organization to minimize risk and insure ROI of investments in software development projects.

Bibliography

Atwood, M.E., Gray, W.D., and John, B.E. (1996). Project Ernestine: Analytic and Empirical Methods Applied to a Real-World CHI Problem. In Rudisill, M., Lewis, C., Polson, P.B. and McKay, T.D. (Eds.). Human-Computer Interface Design (pp. 101-121). San Francisco, CA: Morgan Kaufmann Publishers.

Bias, R. G and Mayhew, D. J., Eds. (1994). Cost Justifying Usability. Boston, MA: Academic Press.

Bias, R. G and Mayhew, D. J., Eds. (2004 - forthcoming). Cost Justifying Usability (2nd Edition). San Francisco, CA: Morgan Kaufmann Publishers..

Card, S.K., Moran, T.P. and Newell, A. (1983). The Psychology of Human-Computer Interaction. Hillsdale, NJ: Lawrence Erlbaum Associates, Publishers.

John, B.E. and Kieras, D.E. (1996). Using GOMS for User Interface Design and Evaluation: Which Technique? In ACM Transactions on Computer-Human Interaction, 3, 287-319.

John, B.E. (1990). Extensions of GOMS Analysis to Expert Performance Requiring Perceptions of Dynamic Visual and Auditory Information. In *CHI '90 Conference Proceedings*, 107-115.