# Tracking the Effectiveness of Usability Evaluation Methods

Bonnie E. John & Steven J. Marks*

12 August 1996
CMU-HCII-96-102

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

*Standard Microsystems Corporation
Irvine, CA 92718

Also appears as Computer Science Technical Report
CMU-CS-96-160

Work on this paper by Bonnie John was supported by the Advanced Research Projects Agency, DoD, and monitored by the Office of Naval Research under contract N00014-93-1-0934. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Office of Naval Research, NCCOSC, the Advanced Research Projects Agency, or the U. S. Government.

Bonnie John is associated with the Human-Computer Interaction Institute and the Departments of Computer Science and Psychology at Carnegie Mellon University.

**Keywords:** usability evaluation methods, Claims Analysis, Cognitive Walkthrough, GOMS, Heuristic Evaluation, User Action Notation, usability engineering

## Abstract

We present a case study that tracks usability problems predicted with six usability evaluation methods (Claims Analysis, Cognitive Walkthrough, GOMS, Heuristic Evaluation, User Action Notation, and simply reading the specification) through a development process. We assess the methods' predictive power by comparing the predictions to the results of user tests. We assess the methods' persuasive power by seeing how many problems led to changes in the implemented code. We assess design-change effectiveness by user testing the resulting new versions of the system. We conclude that predictive methods are not as effective as the HCI field would like and discuss directions for future research.

## 1. Introduction

Usability evaluation methods (UEMs) have been developed that can be used at the specification stage of design (e.g., John & Kieras, in press; Nielsen & Mack, 1994). As these techniques proliferate, developers want to know which they should use for their particular design situations, instructors want to know which they should teach to their students, and researchers want to know which methods need further development and where their research efforts could have the most effect.

To these ends, most empirical work to date has focused on assessing the predictive power of UEMs. That is, does a UEM predict usability problems that users actually encounter? Some techniques have been extensively tested in the laboratory and the field (e.g., GOMS, John & Kieras, in press); others have yet to be systematically evaluated. There have been a few attempts to compare the predictive power of different UEMs with experiments (but methodological flaws make their conclusions suspect, Gray & Salzman, 1996). Case studies also provide information about the predictive power of techniques in specific circumstances (e.g., John & Mashyna, in press; John & Packer, 1995).

Although predictive power is an important aspect of UEMs, there are other aspects that contribute to their effectiveness in a development process. A UEM must express problems, and the evidence for them, in ways that motivate developers to change the code. A UEM should also lead to design changes that actually fix the problem. For most UEMs, research has not produced any measures of these other aspects of effectiveness. The purpose of this paper is to present some results for these other dimensions and examine implications for future UEM research.

### 1.1. An Effectiveness Tree

Figure 1 shows an *effectiveness tree* for predicted usability problems[1]. Any usability problem predicted before a system is built can either occur to users of that system or not (*observed* or *not observed*). This measure is the *predictive power*. Typically, this information is not known in a normal development process, but has been studied by HCI researchers as discussed above.

---

[1] The effectiveness tree was introduced in an unpublished senior thesis (Marks, 1996).

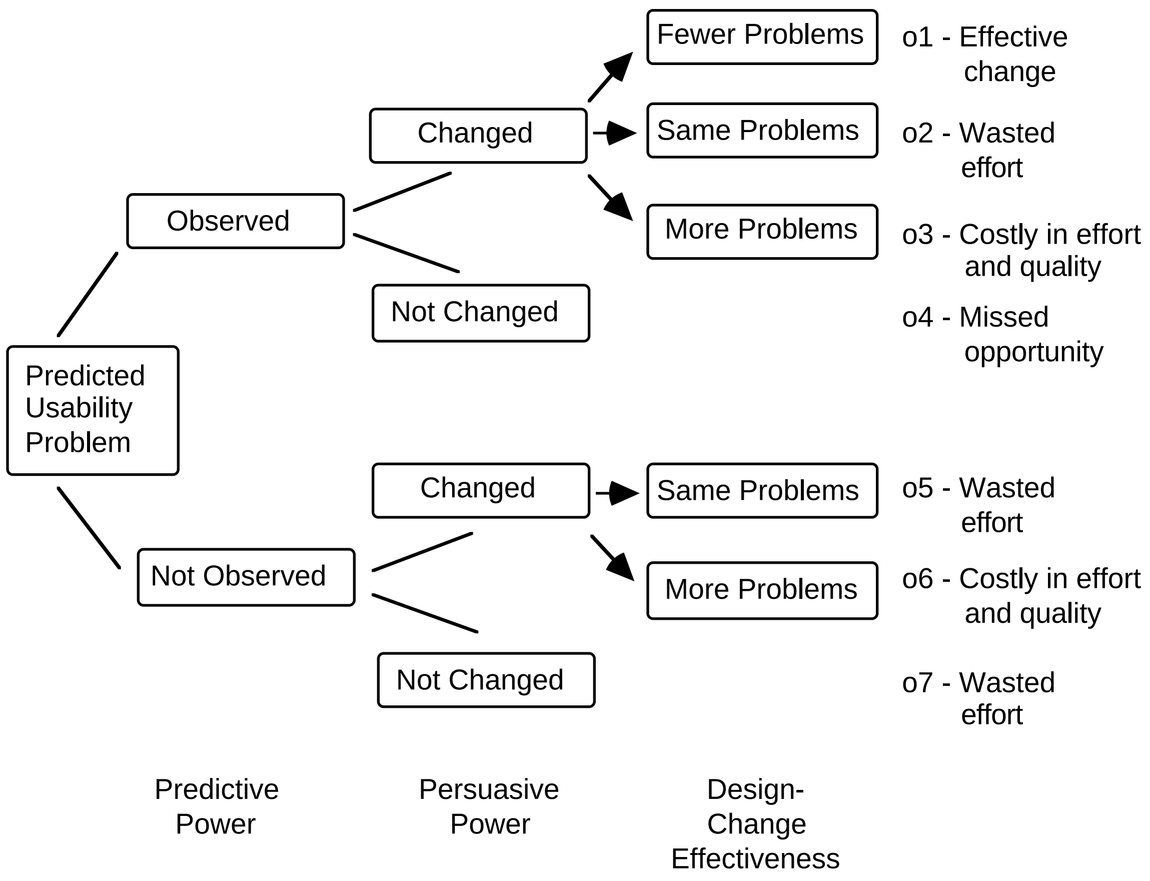


Figure 1. An effectiveness tree.

Any predicted usability problem can either motivate the development team to change the code or not (*changed* or *not changed*).[2] This motivation could have many causes. For instance, a UEM may have demonstrated a high predictive power for similar products and the development team may have come to believe its predictions. Even without a successful history, a UEM may provide evidence for the problem in a form that capitalizes on developers' knowledge and process, making it easy to incorporate. We call this measure *persuasive power*. Other factors besides the persuasiveness of the UEM come into play at this time, e.g., how difficult it is to change the system, release deadlines, and available staff. However, without some degree of persuasiveness, development teams will never expend any effort to change a system they had considered good enough to specify in the first place.

[2] The lines from Observed/Not Observed to Changed/Not Changed in the effectiveness tree have no arrowheads, indicating that they are *not* causally connected. In the normal development process, the development team does not implement an original specification and observe real-world use before deciding to make a change to the original specification. It bases that decision on the evidence provided by a UEM alone.

After deciding to change the system, the development team must decide exactly what changes to make. Some UEMs make design suggestions; some simply identify the problem leaving development teams to conceive of the change. If a UEM suggests a change, the development team decides to adopt it or think of something else. Ultimately, the implemented change may reduce the number of problems users experience, leave performance the same, or introduce more problems than before. We call this measure *design-change effectiveness*.

The most desirable outcome is the top path through this tree (outcome o1). A predicted usability problem would actually be observed in real-world use if not fixed, it provides motivation for the development team to change the code, and the change they implement actually fixes the problem. Disastrous outcomes are o3 and o6, where a predicted usability problem persuades the development team to make a change, but the change introduces usability problems that result in worse performance. This is particularly egregious for o6, where there was no observable problem in the original system. These outcomes take time and effort and result in an inferior product. The other outcomes may be considered less severe. O4 is a missed opportunity to improve the system. O2, o5 and o7 are simply wasted effort.

In the remainder of this paper, we will demonstrate the use of an effectiveness tree by tracking 54 usability problems, predicted with several different UEMs, through a development process. Section 2 presents how predicted usability problems were generated. Section 3 shows how these problems were addressed by development. Section 4 describes our usability tests. Section 5 gives the procedure for tracking the predicted problems. Finally, we present the results, argue for and against their generality, and discuss their implications for future HCI research.

## 2. Predicting Usability Problems[3]

### 2.1. Participants

Six analysts participated in this study. Each analyst chose a UEM based on an introductory lecture on evaluation methods (Butler, Jacob & John, 1994), learned it from the literature, and used it to evaluate a specification of a multi-media authoring tool called the VolumeViewBuilder (henceforth, *Builder*). The analysts received course credit for participating in this case study.

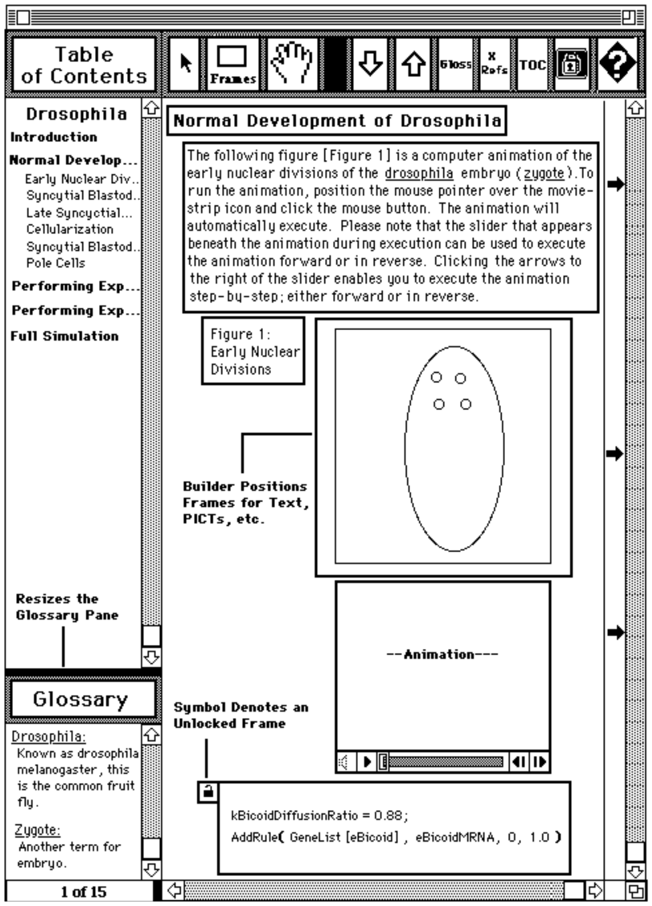
The analyst using Claims Analysis (CA) had a bachelor's degree in electrical engineering and fine art. He was skilled in six programming languages, had commercial experience as a programmer and was an engineering doctoral student at Carnegie Mellon University. The analyst using Cognitive Walkthrough (CW) had a masters in computer science, was skilled in two programming languages, had commercial experience as a programmer, and was a staff programmer in the CS department . The analyst using GOMS had a masters in architecture, was skilled in three programming languages, had commercial experience as a programmer, and was a doctoral student in architecture. The analyst using Heuristic Evaluation (HE) had a masters in English, was skilled in two programming languages, had

[3] More details about this analysis situation can be found in John & Packer, 1995 and John & Machyna, in press. Those papers report data from the CW analyst's diary only, but the analysis situation is the same.

no commercial experience in software development, and was a doctoral student in rhetoric. The analyst using User Action Notation (UAN) had a bachelors in CS, was skilled in three programming languages, had commercial experience as a programmer, had taken an undergraduate HCI course and had corporate training in how to use guidelines, and was a masters student in software engineering. The analyst who used a baseline condition, just reading the specification repeatedly, was skilled in one programming language, was a third-year undergraduate CS major, had no commercial experience, and had recently taken an undergraduate HCI course.

## 2.2. Materials and Procedure

The analysts were given two documents with which to do their analyses: the 35-page user interface specification of the ACSE multimedia authoring system, Figure 2, (Gallagher & Meter, 1993) and a 55-page target multimedia document (Pane, Corbett & John, 1996). They were also given two forms to fill out as they did their analyses: a diary form (adapted from Rieman, 1993) and a problem description report (PDR, adapted from Jeffries, Miller, Wharton & Uyeda, 1991).



The following figure [Figure 1] is a computer animation of the early nuclear divisions of the drosophila embryo (zygote).To run the animation, position the mouse pointer over the movie-strip icon and click the mouse button. The animation will automatically execute. Please note that the slider that appears beneath the animation during execution can be used to execute the animation forward or in reverse. Clicking the arrows to the right of the slider enables you to execute the animation step-by-step; either forward or in reverse.

Drosophila: Known as drosophila melanogaster, this is the common fruit fly.

Zygote: Another term for embryo.

```
kBicoidDiffusionRatio = 0.88;
AddRule( GeneList [eBicoid] , eBicoidMRNA, 0, 1.0 )
```

Figure 2. Example of an illustration in *Volume View Interface Design* (Gallagher & Meter, 1993, p. 16, reprinted by permission). In that document, this is a full-page illustration of the screen of the Builder application.

The analysts worked on their own for an elapsed time of ten weeks, filling out the diaries and PDRs as they worked. They met together (with the first author) for about an hour once or twice a week to discuss the analysis process (not the content of the analyses). That is, they discussed problems getting or understanding papers, problems making the techniques applicable to the Builder, types of information their techniques needed or provided, but not specific usability problems they had found in the Builder. Each analyst produced a stack of PDRs, detailed diaries, and a brief written report.

## 2.4 Specific UEMs

Since these UEMs are evolving, several versions of each method exist in the literature. Therefore, we must be clear about which version each analyst used in their assessment of the Builder's specification.

The CA analyst used both (Kellogg, 1989) and (Carroll & Rosson, 1991) as examples of how to do use the method. He expressed the problems he identified in the "claim -- but" format shown in these papers.

The CW analyst read about several versions of the method, but used the version in (Wharton, Rieman, Lewis & Polson, 1994). He chose tasks based on the target document, derived the correct procedures from the specification, and did his walkthroughs alone. A detailed account of this analyst's experience using CW can be found in (John & Mashyna, in press; John & Packer, 1995).

The GOMS analyst relied primarily on (Kieras, 1994). He specified the goal hierarchy for several tasks derived from the target document and made quantitative predictions of execution and learning time. In addition, Kieras suggests checking the analysis for "naturalness" of the goal hierarchy. This analyst decided that the questions posed by CW were a good means of checking "naturalness" so he kept CW's questions in mind as he examined his GOMS analysis; about 40% of his PDRs were credited to assessing "naturalness" in this way.

The HE analyst used (Nielsen, 1993) as her primary source. She did an HE herself, and recruited three additional evaluators: two with CS backgrounds, one in graphic design. She trained them with (Molich & Nielsen, 1990), and recorded all the problems they identified and suggestions they made. This analyst did not remove duplicate problems from the set of raw PDRs. Since HE with several evaluators typically has duplicate problems, the first author and a colleague examined the raw PDRs, came to a consensus about duplicates, and removed them. The duplicates were 5% of the total raw PDRs, lower than we expected, and lower than depicted in other HE writings (e.g., Nielsen & Mack, 1994: p. 27).

The UAN analyst found (Siochi & Hartson, 1989) best for understanding the basics of UAN, but used the more detailed (Hartson, Siochi & Hix, 1990) as a guide when doing the analysis itself. This analyst understood that the goal of UAN is to write an unambiguous specification, not to point out usability problems. However, she wanted to see if the process of formalizing a prose specification would have the side-effect of identifying usability problems or design improvements as well.

The analyst who just read the specification repeatedly did not use any particular method. He was instructed to note in the diaries if he noticed himself using any particular method in his reading, but he did not write down any known UEM. He did record that he read the specification once through deliberately looking only at whether commands were undoable or not.

## 2.5 Results of the UEM analyses

The results of the UEM analyses can be characterized by the usability problems they predicted. Other measures might also be included in other analyses of these rich diaries, e.g., the effort exerted by the analysts, in what activities they spent their time, their doubts and insights about the UEMs, but an analysis of those data is beyond the scope of this paper (but see John & Mashyna, in press; or John & Packer, 1995, for detailed analyses of the CW case).

323 raw PDRs were generated by the analyses: 24 by CA, 42 by CW, 44 by GOMS, 127 by HE, 17 by UAN and 69 by Reading (Table 1). Most of the analysts attributed the detection of these problems directly to their UEM, as opposed to having found them by reading the specification or other personal judgment. A notable standout is that the analyst using UAN did not attribute any PDRs directly to the UAN technique itself, but called 94% of the PDRs a "side effect" of the technique. As discussed above, this is not a surprising result, as the goal of UAN is to produce an unambiguous specification, not to identify usability problems.

Table 1. PDRs generated by the analysts and the sources to which they attributed the PDRs.

| UEM | Raw PDRs | number from UEM | percent from UEM | number side effect | percent side effect |
|---|---|---|---|---|---|
| CA | 24 | 19 | 79% | 0 | 0% |
| CW | 42 | 26 | 62% | 4 | 10% |
| GOMS | 44 | 32 | 73% | 0 | 0% |
| HE | 127 | 111 | 87% | 2 | 2% |
| UAN | 17 | 0 | 0% | 16 | 94% |
| Reading | 69 | N/A | N/A | N/A | N/A |
| TOTAL | 323 | 188 | 58% | 23 | 7% |

## 3. Addressing the Predicted Usability Problems

Much of the Builder had been implemented according to the specification, independent of the UEM analyses. Some changes were made during implementation, by agreement within the development group.

Given an implementation and a set of predicted problems, decisions must be made about what code to change. In the real world, the development team as a whole could make decisions, management or an external customer could set priorities, or individual developers could decide to change the code under their control. These procedures depend on the size and complexity of the product and the organizational culture. In our case, the situation was as follows.

### 3.1 Development Participant

The ACSE project (which included the Builder) and its predecessors had been in existence for more than 15 years at Carnegie Mellon University, producing several versions of novice programming environments currently used in universities across the Unites States (Miller, Pane, Meter & Vorthmann, 1994). The project was winding down at the time of our involvement; one full-time developer, *Dev1*, was available to assess the PDRs and fix the code.

Dev1 had worked on these systems for almost 10 years. Although he was not an author of the specification, he contributed to it and had written much of the code. Dev1 had taken an HCI course several years prior to this work and had classroom knowledge of all these UEMs.

### 3.2 Materials and Procedure

We gave Dev1 the PDRs, transcribed, printed one to a page, and grouped by UEM that produced them.[4] These included the prose description of the problem, the analyst's judgments of frequency and severity, and the source of their judgments. Dev1 handled the PDRs in the following order: CW, UAN, CA, Reading, GOMS, HE.

Dev1 recorded his reasoning about why to fix a problem or not on the PDR, extended with additional questions for that purpose. Dev1 first judged whether a PDR reported an actual usability problem of the Builder, a problem with the specification itself, or a problem with the target document which did not reflect a usability problem with the Builder.

[4] A clerical error, not found until after all systems had been built and all usability tests completed, prevented 32 HE PDRs from being given to the developer. A post-hoc analysis of these PDRs showed them to be very similar to the other HE PDRs. They spanned all aspects of the Builder, 4 were in portions of the Builder that are not yet implemented, 7 did not exist in the current implementation because of the changes introduced in the normal development process, 4 were problems with the specification or analyst's misreading of the specification, 4 were duplicates of other UEMs' PDRs which had been passed along to the developer, leaving 13 which might have influenced the design. Dev1 was kind enough to rate these PDRs after the fact, and estimated that he would have changed code because of 2 of them, but these changes were exactly what he actually implemented for the HE version. Therefore, to our best knowledge the HE version would not have been changed due to these extra PDRs and only the 88 HE PDRs actually passed to Dev1 were tracked.

If Dev1 interpreted a PDR as an actual usability problem, he recorded whether this problem existed in the current implementation of the Builder. If so, he recorded how important he thought it was to fix the problem on a five-point scale (1=not important, 5=imperative). He recorded alternative ways to fix the problem and rated each on a five-point scale of difficulty (1=trivial to fix, 3= 1 person/week, 5 = almost impossible to fix). Finally, Dev1 recorded his final decision to fix the problem or not, which option to use if there were alternatives, and a reason for that decision.

Dev1 then changed the code of the baseline implementation to make a new version of the Builder. If a predicted usability problem had already been fixed when handing an earlier UEM, he simply included the new code in that UEM's version of the Builder. Dev1 worked on these changes to the code full time for about ten weeks.

## 3.3 Results and Discussion of Development Participation

Of the 284 PDRs passed to Dev1, he thought two contained two problems each, so he filled out questionnaires for 286 predicted usability problems (Table 2). Of these, Dev1 believed that 56 (20%) were not actually usability problems with the Builder. UAN stands out as having a high percentage of non-problem PDRs (82%) and all of these are problems with the specification. This result is not surprising, as UAN is a technique for disambiguating a specification. CW, GOMS and Reading reported relatively few non-usability problems (5, 13 and 6%, respectively, all specification-related). Of the 20% non-usability problems reported by CA, one third (2 PDRs) were with the specification, one was a criticism of animation as a learning tool based on reading the analyst had done, half (3) were introduced when the analyst used the technique iteratively, criticizing his own design suggestions which were not part of the original design. About two-thirds of the HE non-usability problems were attributed to the specification; about one-third to the target document. All of the target-document problems came from the first HE analyst, an English graduate student concentrating in rhetoric. Two non-usability problems, both generated by the HE analyst with a CS education, were based on assumptions about memory requirements of the Builder that Dev1 felt were incorrect. Perhaps these non-usability non-specification problems reflect the analysts' background more than HE itself. 230 predicted usability problems remained to be analyzed further.

All of the analysts found problems with parts of the Builder which were not yet implemented (NYI). Table 3 shows that CA, CW, and Reading spent 1/4 to 1/3 of their effort in these areas. (UAN's 100% in these areas is probably an artifact of the small number of predicted usability problems.) 70% of GOMS's 17 NYI problems were with the procedure for creating cross-references, possibly reflecting the emphasis that GOMS puts on procedures and the difficulty of that particular procedure. Only 5% of HE's problems were in NYI areas and we do not have an explanation for this result.

Table 3 also shows the number of predicted usability problems that had changed during the normal development process, without the benefit of these UEMs. Most of the techniques cluster around 30%. CA is lower, at about 16%. (UAN's 0% is probably an artifact of its low number of predicted usability problems.)

Problems that do not exist in the baseline implementation of the Builder cannot possibly be observed in think-aloud usability tests, therefore, we dropped them from further analysis. This left 100 potentially observable predicted problems.

Table 2. Dev1's judgment of the PDRs passed to him.

| UEM | Number PDRs passed to Dev1 | number problems addressed by Dev1 | number NOT usability problems | percent NOT usability problems | number of usability problems |
|---|---|---|---|---|---|
| CA | 24 | 25 | 6 | 24% | 19 |
| CW | 42 | 42 | 2 | 5% | 40 |
| GOMS | 44 | 45 | 6 | 13% | 39 |
| HE | 88[3] | 88 | 24 | 27% | 64 |
| UAN | 17 | 17 | 14 | 82% | 3 |
| Reading | 69 | 69 | 4 | 6% | 65 |
| TOTAL | 284 | 286 | 56 | 20% | 230 |

Table 3. Predicted usability problems in areas that were not yet implemented (NYI), or changed by the development team in the normal course of implementation. The numbers in this figure do not add up to the numbers in Table 2's last column because both CW and HE had one problem which Dev1 said did not exist in the current implementation without explanation, and GOMS had one problem for which Dev1 provided a simple "work-around" obviating the problem in his opinion.

| UEM | number of usability problems | number NYI | percent NYI | number changed during development | percent changed during development | problems still existing |
|---|---|---|---|---|---|---|
| CA | 19 | 5 | 26% | 3 | 16% | 11 |
| CW | 40 | 13 | 33% | 11 | 28% | 15 |
| GOMS | 39 | 17 | 44% | 11 | 28% | 10 |
| HE | 64 | 3 | 5% | 19 | 30% | 41 |
| UAN | 3 | 3 | 100% | 0 | 0% | 0 |
| Reading | 65 | 20 | 31% | 22 | 34% | 23 |
| TOTAL | 230 | 61 | 27% | 66 | 29% | 100 |

Table 4 shows the number of changes to the code. Neither CA nor UAN resulted in any changes. The other techniques persuaded Dev1 to make between 2 and 7 changes to the code. HE and Reading had the highest *compression ratio*, where an average of more than three problems could be fixed with a single code change, whereas GOMS had a one-to-one mapping between the predicted problems which led to change and the change itself. Interestingly, the compression ratio does not account for any of the variance in the percentage of problems that led to a change, indicating that multiple reasons to make a change may be no more persuasive than one well-supported reason.

Table 4. Predicted usability problems that persuaded Dev1 to change the code.

| UEM | problems still existing | number led to code change | percent led to code change | number of changes in code |
|---|---|---|---|---|
| CA | 11 | 0 | 0% | 0 |
| CW | 15 | 11 | 73% | 5 |
| GOMS | 10 | 3 | 30% | 3 |
| HE | 41 | 7 | 17% | 2 |
| UAN | 0 | 0 | 0% | 0 |
| Reading | 23 | 9 | 39% | 7 |
| TOTAL | 100 | 30 | 30% | |

## 4. Usability Testing

The next stage in tracking predicted usability problems is to collect data about their occurrence with users. Ideally, we would deploy the Builder and observe real users doing their actual jobs. However, that was not possible and we approximated the ideal situation with think-aloud usability tests in the laboratory. Such testing is heavily used in industry and "may be the single most valuable usability engineering method" (Nielsen, 1993: p. 195).

### 4.1 Tasks

We gave our users five tasks, all within the use-scenario that they were creating course material for a biology class. These tasks were constructed from an actual biology lesson built with an earlier version of the Builder (Pane et al., 1996). We included creation and modification tasks and exercised the major features of the Builder (text, graphics, animation, code, table of contents, glossary, bookmarks, etc.). The task set included some procedures we had failed to exercise in a previous study (John & Mashyna, in press). As a pragmatic constraint, we restricted these tasks so that all training, data-collection, and debriefing could be accomplished in 2 hours.

The first task was to create a three-page multimedia document from hard-copy, attached to an science simulation. This involved cutting and pasting text, pictures and an animation from other Macintosh applications, and adding entries into the table of contents and glossary. The second task was to switch page 2 with page 3. The third task was to add another page, enter another glossary entry, and modify the definition of an existing entry. The fourth task was to add a final page that included a code fragment and to ensure that future users of the document could not change that code. The last task was to save the document in an editable version for the professor and in an exercise version for the students.

### 4.2 Participants

Twenty business undergraduates participated in the usability tests, four[5] for each of the five versions of the Builder. All had experience with the Macintosh, were familiar with Microsoft Word, and knew how to cut and paste between applications. The students earned extra credit toward class grades for participating in the study. Each participant was randomly assigned to a version of the Builder, in blocks.

### 4.3 Apparatus

The tests used a Macintosh Quadra with a 17 inch color monitor, running System 7.5. MSWord5.1 held source text and pictures; MoviePlayer1.0 held source animations. Five versions of the Builder were used: vBL1.2 (baseline), vCW1.2 (built from Cognitive Walkthrough suggestions), vGM1.0 (GOMS suggestions), vHE1.0 (Heuristic Evaluation suggestions), and vRD1.1 (Reading suggestions). All sessions were videotaped.

### 4.4 Procedure & Results

Each participant was given simple skills assessment tasks to demonstrate his or her proficiency with the Macintosh. The participant had to switch between two applications, copy a graphic from one application to another, and, after a brief training, copy an animation from one application to another. The participant was trained how to do a think-aloud verbal protocol and practiced thinking aloud while playing solitaire.

The participant was then shown a program on the computer screen and told that the Builder allowed professors to attach course material to programs. This program was a simulation of fruit fly embryo development. They were given hard-copy and asked to create a volume

[5] Prior research has shown that using three or four participants in a usability test maximizes the ratio of the number of usability problems found to the effort involved in running the test (Nielsen & Landauer, 1993; Virzi, 1990; Virzi, 1992). However, this number is still the subject of debate and can be much higher if the probability of finding a problem in the interface is very low (e.g., see (Lewis, 1994)).

that looked like these pages and attach it to the program. They were told that source text, pictures and an animation were in other applications already open on their screen.

The participants were not given any specific instructions about the Builder. This instructionless learning paradigm represents how many people begin using new applications (Carroll & Rosson, 1987).

After completing the first task, the participant was asked to do the second task; after that, the third, and so on until all five tasks were done. To keep the tasks reasonably short, we enforced two time limits. First, if a participant articulated a goal and failed to make any progress toward that goal for three minutes, the experimenter showed the participant how to accomplish that goal. Second, each task was assigned a maximum duration by quadrupling the time the second author could perform the task. These durations were checked with three pilot users. If the maximum duration was reached, the experimenter stopped the participant and asked the participant to move onto the next task.

This procedure resulted in 20 videotapes, four for each version of the Builder. These videotapes are the data from which to track the problems described in the next section.

## 5. Problem Tracking

### 5.1 Problems to track

We had to establish which predicted problems were potentially observable in the usability tests. Even though the suite of tasks was created to cover system features, the usability analysts could predict problems that would never show up in the laboratory environment we provided. If we were to track these problems, it would give a distorted measure of predictive power for the UEMs used.

First, many predicted problems concerned the quality of course materials rather than the Builder itself. For instance, there were several suggestions about making pages easier to read (e.g., constraining the amount of text on a page, distributing white-space evenly, etc.) These problems could only be observed with end-users (students), measuring their understanding of the course content or the time it takes them to come to a threshold of understanding. Other problems would arise only if the participant were the actual author of the curriculum. That is, only the true author would need to do things like compare two alternative layouts; our participants were simply trying to construct an electronic version of the hard-copy we gave them. Finally, some predicted problems were with features that the task suite failed to address, despite our careful selection. For instance, we did not ask the participants to edit a code frame. For these reasons, we judged 33 predicted problems to be unobservable in our usability tests.

We then combined problems that would be indicated by the same behavior. For instance, the analyst using CA said "People may erroneously rely on having the undo option available when it is, in fact, inactive" and the Reading analyst said "In general, all commands should be undoable by choosing 'undo'." Although not precise duplicates (i.e., CA's problem could be fixed by warning the user that undo is inactive), the behavior indicating the problem involves the user trying to use the undo command when it isn't active. Combining similar problems reduced our list to 54 problems (Figure 4).

## 5.2 Procedure and Results

The second author annotated the 20 videotapes using the MacSHAPA observational analysis tool (Sanderson et al., 1994). Using the list in Figure 4, he specifically looked for instances of the 54 predicted difficulties. He adhered to a set of criteria for reporting that a problem was observed (Figure 3). A second analyst was trained on the criteria with three problems in each of two videotapes, with feedback on her categorizations. After training, she analyzed 4 tapes, looking for 11 different problems, for a total of 38 ratings, and had a 82% absolute agreement with the second author's ratings on whether the problems were observed. A consensus opinion was reached for each of these ratings. The first analyst had a 92% agreement with the consensus; the second, 89%. Since this inter-rater reliability is good, each rater did a distinct portion of the remaining problems.

The user articulated a goal and cannot succeed in attaining that goal within 3 minutes (then the experimenter steps in and shows him or her what to do).

The user articulates a goal, tries several things and then explicitly gives up.

The user articulates a goal and has to try three or more things to find the solution.

The user does not succeed in a task. That is, when there is a difference between the hard-copy document the user was given and the Volume the user produced.

The user expresses surprise.

The user expresses some negative affect or says something is a problem.

The user makes a design suggestion.

Figure 3. Criteria for reporting that a usability problem was observed in a videotaped think-aloud usability test.

It is important to note that we used a different procedure than work comparing the problems predicted by a UEM to those observed in a usability study. That work typically looks for *all* problems that a user has, rather than concentrating on verifying predicted problems (e.g., John & Mashyna, in press). Our procedure is different because our goal is to track predicted usability problems through the development process rather than to compare techniques. Our procedure reliably detects whether a predicted problem is experienced by users but it will not discover usability problems which the UEM does not predict.

The results of the videotape analysis are presented in Figure 4. A one-line summary of each problem gives a sense of the content and range of the predictions. The sources of each prediction are listed in column 3. All 54 problems were tracked in the videotapes of the baseline version, indicated by a number appearing in every row of that column. Problems were tracked in the other versions only if a code change was made attempting to fix the problem. A number in the last four columns indicates that a problem was tracked in that version; a dash indicates that it was not.

Insert Figure 4, page 1 here

Insert Figure 4, page 2 here

## 6. Discussion and Conclusions

Given the predicted problems, Dev1's judgments of those problems, and the user tests, we can fill in the effectiveness tree for this development case. Before discussing the tree and its implications, we will present the limitations of our study to qualify the conclusions we draw from the tree. We also lay out some directions for future research.

### 6.1 Strengths and Limitations of the Data

Many aspects of this study may generalize to real-world use of UEMs, but others limit generalization.

*The application and specification.* These UEMs were developed in the era where the paradigmatic human-computer interaction involved an office worker in front of a PC. Since the Builder fits in this paradigm it may be more suited to these UEMs than other applications involving virtual reality, for instance, or collaborative work. The specification used prose and figures. Although such specifications are widely used, these results may not generalize to other forms, like formal specifications or prototype-as-specification.

*The problem prediction process.* Since this work follows a case-study paradigm, not an experimental one, we had a single analyst using each UEM. It is difficult to attribute credit or blame to the UEM itself as opposed to the analyst's background without deeper analysis of the diary data produced in the study (e.g., see John & Mashyna, in press). Therefore, we will not make strong comparisons between techniques, but only look for dominant patterns in the data.

These analysts were novices at the techniques. While novice analysts are not uncommon in real-world practice (Dillon, Sweeney & Maguire, 1993), these data do not reflect the best work of analysts expert in a particular UEM.

These analysts were not developers of the system they were critiquing. Although many companies bring in usability analysts as consultants, many other projects include usability analysts as full team members. It is unclear whether additional knowledge of the underlying system would produce less naive predictions or fewer criticisms because of the investment the analysts themselves have made in the design.

Finally, the problems were generated with specific versions of the UEMs (section 2.4). Most of these methods are under development and the performance displayed here may not exist in newer versions of the technique.

*The development involvement.* There was a single developer making all the judgments and all the code changes, under a deadline. As a major contributor to the design and the code, Dev1 therefore had the same background as programmers on other real-world development projects. However, he was working alone on these changes after development on the project had essentially stopped. It is unclear whether more programmers, or more time, would have produced a large quantity or higher quality design changes. In addition, Dev1 did not have the opportunity for dialog with the analysts. In a situation where the analysts are part of the design team, perhaps these techniques would be more persuasive.

*The usability tests.* Our usability tests have the same limitations as many conducted for real-world products. Our tasks were short so that the tests could be of reasonable duration. Our participants had similar computer experience as target users, but were not domain experts or actual authors of educational materials. There was no extensive instruction or practice with the system. More training and longer tasks might have uncovered problems that only come with scale or expertise (for instance, the differences in expert performance time that GOMS is designed to predict, e.g., Gray, John & Atwood, 1993; John & Kieras, in press). In addition, more users might have uncovered more problems in general (Lewis, 1994; Virzi, 1992).

## 6.2 Effectiveness tree

Figure 5 shows the effectiveness tree for our data. The numbers in each box indicate how many predicted problems are in the category named by the box and the path to it. The UEMs listed predicted the problems in each box. For instance, the box at the top left says that 6 problems predicted by CW, HE or Reading, were observed in the baseline user tests, led to changes in the code, and fewer people displayed those problems in the user tests of the revised Builders.

Slightly less than half of the 54 problems were observed in the usability tests. Of those, 50% led to changes in the code, and about one half of those changes actually resulted in fewer observations of users having troubles. Another quarter had the same number of users with problems and the last quarter had more observations of users with difficulties than the baseline version of the Builder. Of the predicted problems that were not observed, about one quarter led to changes in the code, all of which produced the same number of user problems in the revised Builder as in the original Builder.

Because of the limitations of our study discussed above, we focus on big patterns in these data rather than small differences. For example, UAN is totally absent from this chart. Although the analyst using UAN predicted some usability problems as a side effect of the technique, none survived in the tracking process to this stage; most were identifying problems in the specification. This is evidence that UAN (circa 1990) is not a technique for finding usability problems, but for disambiguating a specification, as advertised.

A striking pattern is that relatively few of the predicted problems ended in the most desirable outcome, o1 - Effective change. Only 11% of these problems were observed and led to code changes that gave fewer users difficulty. Half that number ended in outcome o4, which took considerable coding effort and resulted in an inferior system. This is evidence that the UEMs studied here are not as effective as the HCI field would like and we have a dire need for more research into UEMs.

Other patterns suggest that the situation is not hopeless. First, no problems ended in o6, an outcome that also results in an inferior system. Second, 80% of the problems that were not observed also were not changed (o7), a higher percentage than observed problems that did not lead to code changes. Perhaps there was something about these problems, or the way they were presented, that led the developer to decide not to change the code. If this "something" could be discovered and codified, it could be incorporated into the UEMs to reduce these false alarms.

Another pattern is that most outcomes are populated by problems from most of the UEMs. All UEMs predicted some problems that were observed and some that were not. All but

CA persuaded the developer to change the code sometimes and not others. The numbers are too small in the design-change effectiveness column to be confident in specific results (i.e., we would NOT want to conclude that HE never produces worse designs, GOMS never produces better designs, and CW and Reading always produce both better and worse). However, the fact that no design-change effectiveness outcome is populated by a single UEM suggests that all techniques have strengths that further research could mine and limitations that it could reduce. Also, just reading the specification repeatedly without using any particular UEM populates the outcomes as much as do the more structured UEMs.

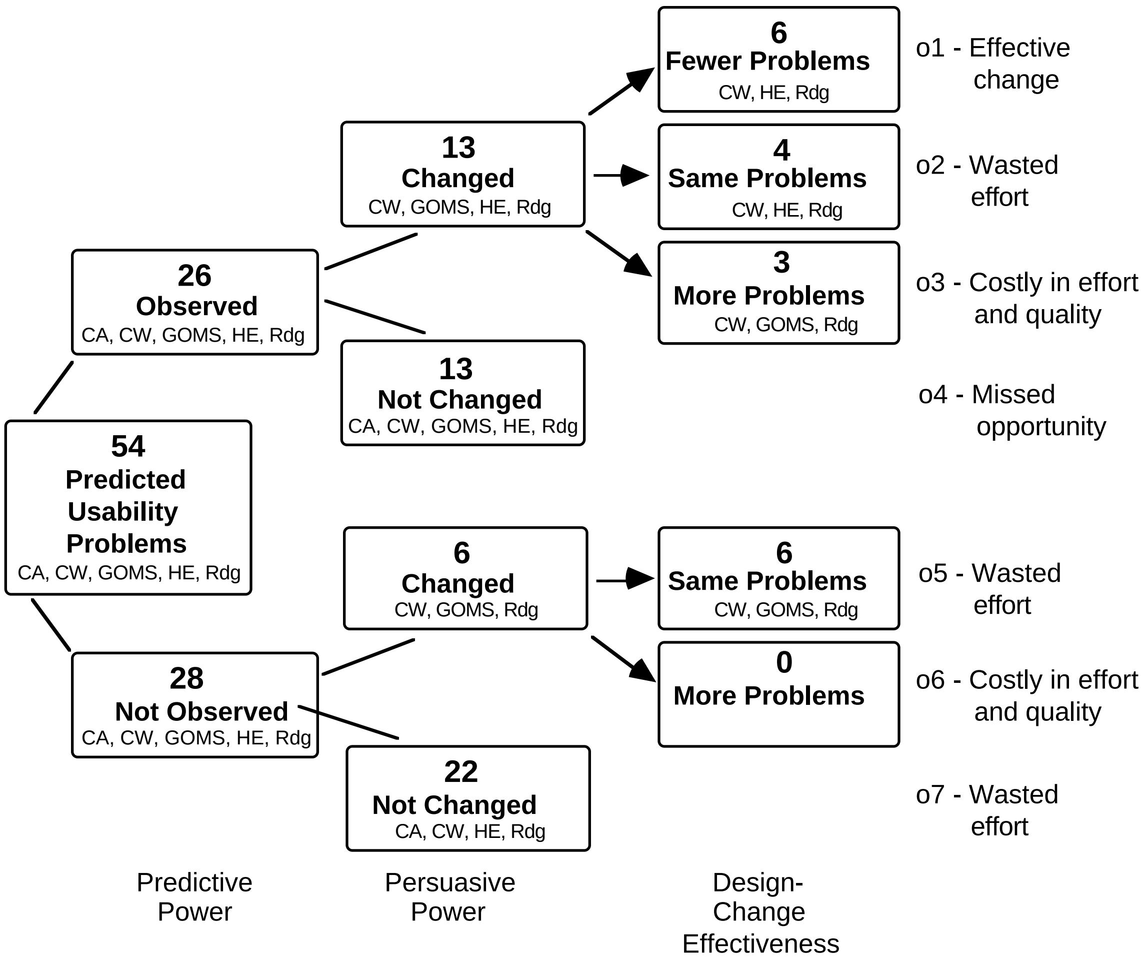


Figure 5. The effectiveness tree for our data.

Figure 6 shows the analysts' average frequency and severity ratings for each problem category in the effectiveness tree, as well as Dev1's average ratings of importance of the problems and difficulty of changing the code. The number of design suggestions made by the analysts and number of those suggestions actually implemented by Dev1 are also shown. These numbers are descriptive statistics only; this study does not have sufficient

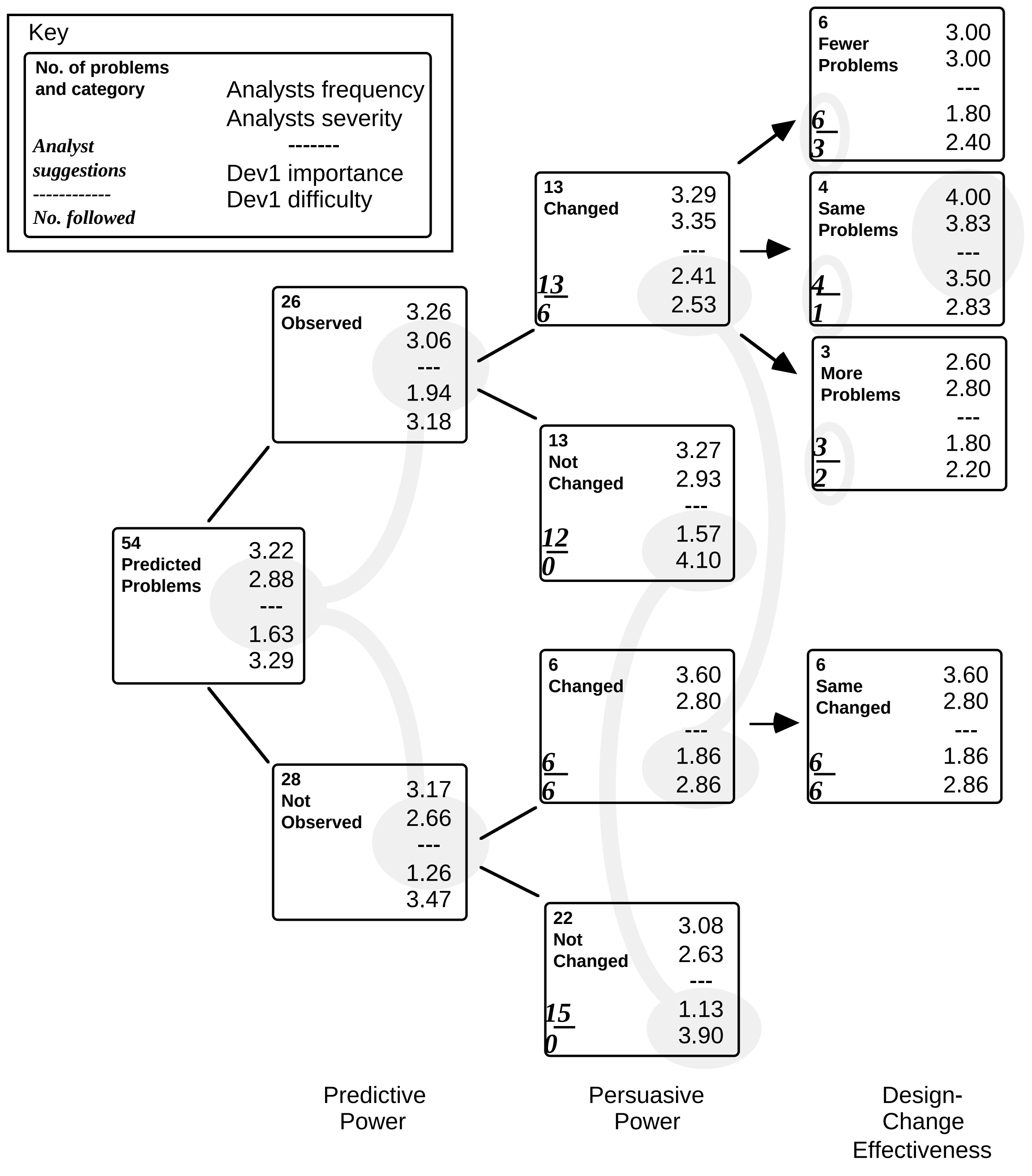

Key
No. of problems and category
Analysts frequency
Analysts severity
-------
Dev1 importance
Dev1 difficulty
Analyst suggestions
------------
No. followed
54 Predicted Problems 3.22 2.88 --- 1.63 3.29
26 Observed 3.26 3.06 --- 1.94 3.18
28 Not Observed 3.17 2.66 --- 1.26 3.47
13 Changed 3.29 3.35 --- 2.41 2.53 13/6
13 Not Changed 3.27 2.93 --- 1.57 4.10 12/0
6 Changed 3.60 2.80 --- 1.86 2.86 6/6
22 Not Changed 3.08 2.63 --- 1.13 3.90 15/0
6 Fewer Problems 3.00 3.00 --- 1.80 2.40 6/3
4 Same Problems 4.00 3.83 --- 3.50 2.83 4/1
3 More Problems 2.60 2.80 --- 1.80 2.20 3/2
6 Same Changed 3.60 2.80 --- 1.86 2.86 6/6
Predictive Power
Persuasive Power
Design-Change Effectiveness

Figure 6. The effectiveness tree with analyst's average frequency and severity ratings, Dev1's average importance and difficulty ratings and the number of analyst's design suggestions made and implemented.

statistical power o produce significant inferential results. Therefore, we again look only at the biggest trends (shaded).

In general, Dev1's importance rating follows the analysts' severity ratings, though on a lower scale. For instance, the analysts and Dev1 all rate the observed problems slightly more important (or severe) than the unobserved problems.

The shaded items in the persuasiveness column show that the problems which led to code changes were more important and less difficult, on average, than the problems Dev1 did not choose to redesign. This may be an indication of wisdom on Dev1's part, or a form of cognitive dissonance (i. e., he changed it, so it must have been more important).

In the final outcomes, the problems that were most important to fix led to solutions that did not improve performance. This does not seem to be due to Dev1 not following the analysts' design suggestions, however. When he did follow their suggestions, performance both improved and got worse (open circles). Again, these numbers are descriptive, not inferential, so these trends should be viewed as hypotheses to test with further research rather than conclusions to apply in practical design environments.

### 6.3 Lessons learned and future work

Despite the cautions in section 6.1, we believe there are still several lessons that we can learn from this work.

*These UEMs are not as effective as HCI would like.* If they were highly effective, there would be a much higher percentage of problems resulting in an effective change, instead of populating the outcomes so evenly.

*There is no particular UEM to blame for this result.* All UEMs populated most of the outcomes, with two exceptions. UAN (circa 1990) did not predict trackable usability problems, but it was not intended as a tool for prediction. CA (circa 1991) did not lead to any design changes. We speculate that its "claim...but" format, presenting both advantages and disadvantages of a design, makes it difficult to see a clearly better way to implement the system.

*Just reading a prose specification many times seems to be as effective as more structured UEMs at detecting and fixing novice-user problem.* This reading might require someone not involved in the actual design, as was our Reading analyst.

All of the lessons of this experience should be tested with further research. The HCI field needs to find productive partnerships between research and development to do actual experiments on UEM use. We need real specifications, real products, real development teams and real users, in sufficient number, to have generalizable results. Other fields, like medicine and education, have well-established methodologies to assess processes; we need to commit the time and resources to follow their lead.

Finally, the effectiveness tree presented here provides an interesting framework for evaluating UEMs more broadly than simple predictive power. However, this framework needs to be exercised on more data and expanded to include usability problems missed by analytic UEMs. In addition, quantitative measures of predictive power, persuasive power and design-change effectiveness need to be established so that many studies can produce comparable data.

## Acknowledgment

We would like to thank the anonymous *Dev1* for his efforts, and Matt Mashyna for his continuing support.

## References

Butler, K. A., Jacob, R. J. K., & John, B. E. (1994). Introduction and Overview of Human-Computer Interaction, *Companion Proceedings of ACM CHI'94 Conference on Human Factors in Computing Systems*, (Vol. 2, pp. 351-352). New York: ACM.

Carroll, J. M., & Rosson, M.-B. (1987). The paradox of the active user. In J. M. Carroll (Ed.), *Interfacing thought*, . Cambridge, MA: MIT Press.

Carroll, J. M., & Rosson, M. B. (1991). Deliberated Evolution: Stalking the View Matcher in Design Space. *Human-Computer Interaction, 6*(3-4), 281-318.

Dillon, A., Sweeney, M., & Maguire, M. (1993). A Survey of Usability Engineering Within the European IT Industry -- Current Practice and Needs, *Proceedings of the HCI'93 Conference on People and Computers VIII*, (pp. 81-94).

Gallagher, S., & Meter, S. (1993). *Volume View Interface Design* : School of Comptuer Science, Carnegie Mellon University.

Gray, W. D., John, B. E., & Atwood, M. E. (1993). Project Ernestine: Validating a GOMS Analysis for Predicting and Explaining Real-World Task Performance. *Human-Computer Interaction, 8*(3), 237-309.

Gray, W. D., & Salzman, M. C. (1996). Damaged merchandise? A review of experiments that compare usability evaluation methods. *Manuscript submitted for publication.*

Hartson, H. R., Siochi, A. C., & Hix, D. (1990). The UAN: A user-oriented representation for direct manipulation interface designs. *ACM Transactions on Information Systems, 8*(3), 181-203.

Jeffries, R., Miller, J. R., Wharton, C., & Uyeda, K. M. (1991). User interface evaluation in the real world: A comparison of four techniques, *Proceedings of CHI, 1991 (New Orleans, Louisiana, April 28 - May 2, 1991)*. New York: ACM.

John, B. E., & Kieras, D. E. (in press). Using GOMS for user interface design and evaluation: Which technique? *ACM Transactions on Computer-Human Interaction*.

John, B. E., & Mashyna, M. M. (in press). Evaluating a multimedia authoring tool with cognitive walkthrough and think-aloud user studies. *Journal of the American Society of Information Systems*.

John, B. E., & Packer, H. (1995). Learning and using the Cognitive Walkthrough Method: A case study approach. *Proceedings of CHI, 1995 (Denver, Colorado, May 7-11, 1995)*, (pp. 429-436). New York: ACM.

Kellogg, W. A. (1989). *Qualitative artifact analysis* (Research Report RC 15768): IBM T. J. Watson Research Center.

Kieras, D. E. (1994). *A guide to GOMS task analysis (Spring 1994)* : University of Michigan.

Lewis, J. R. (1994). Sample sizes for usability studies: Additional considerations. *Human Factors, 36*(2), 368-378.

Marks, S. J. (1996). *Exploratory evaluation of usability inspection technques.* Unpublished Undergraduate Senior Thesis, Carnegie Mellon University.

Miller, P. L., Pane, J. F., Meter, G., & Vorthmann, S. (1994). Evolution of Novice Programming Environments: The Structure Editors of Carnegie Mellon University. *Interactive Learning Environments, 4*(2), 140-158.

Molich, R., & Nielsen, J. (1990). Improving a human-computer dialog. *Communications of the ACM, 33*(3), 338-348.

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

Nielsen, J., & Landauer, T. K. (1993). A mathematical model of the finding of usability problems, *Proceedings of INTERCHI, 1993 (Amsterdam, April 24-April 29)*, (pp. 206-213). New York: ACM.

Nielsen, J., & Mack, R. L. (Eds.). (1994). *Usability Inspection Methods*. New York: John Wiley.

Pane, J. F., Corbett, A. T., & John, B. E. (1996). Assessing dynamics in computer-based instruction, *Proceedings of CHI, 1996 (Vancouver, BC, April 14-18, 1996)*, (pp. 197-204). New York: ACM.

Rieman, J. (1993). The diary study: A workplace-oriented research tool to guide laboratory efforts, *Proceedings of INTERCHI, 1993 (Amsterdam, The Netherlands, 24 April - 29 April, 1993)*, (pp. 321-326). New York: ACM.

Sanderson, P. M., Scott, J. P., Johnston, T., Mainzer, J., Wanatabe, L. M., & James, J. M. (1994). MacSHAPA and the enterprise of Exploratory Sequential Data Analysis (ESDA). *International Journal of Human-Computer Systems, 41*(5), 633-681.

Siochi, A., & Hartson, H. R. (1989). Task oriented representation of asynchronous user interfaces, *Proceedings of CHI, 1989 (Austin, Texas, April 30 - May 4, 1989)*, (pp. 183-188). New York: ACM.

Virzi, R. A. (1990). Streamlining the design process: Running fewer subjects, *Proceedings of the Human Factors Society 34th Annual Meeting*, (pp. 291-294). Santa Monica, CA: Human Factors Society.

Virzi, R. A. (1992). Refining the test phase of usability evaluation: How many subjects is enough? *Human Factors, 34*(4), 457-468.

Wharton, C., Rieman, J., Lewis, C., & Polson, P. (1994). The Cognitive Walkthrough Method: A practitioner's guide. In J. Nielsen & R. L. Mack (Eds.), *Usability Inspection Methods*. New York: John Wiley & Sons, Inc.