MODELING HUMAN ERROR FOR EXPERIMENTATION, TRAINING, AND ERROR-TOLERANT DESIGN

Scott D. Wood
Soar Technology
Ann Arbor, Michigan

David E. Kieras
University of Michigan
Ann Arbor, Michigan

ABSTRACT

Human error in computer systems has been blamed for many military and civilian catastrophes resulting in mission failure and loss of money and lives. However, the root cause of such failures often lies in the system's design. A central theme in designing for human-error tolerance is to build a multi-layered defense. Creating such a robust system requires that designers effectively manage several aspects of erroneous system usage: prevention, reduction, detection, identification, recovery, and mitigation. These also correspond to discrete stages before and after error occurrence where different defensive measures can be taken. Human error models can be used to better understand these stages, the underlying cognitive mechanisms responsible for errors, and ultimately how to design systems and training to reduce the effects of inherent human limitations.

This paper presents a general framework for human error recovery based on five key stages of erroneous performance: the commission of an error, its detection, identification, and correction, and resumption of the original task. These stages constitute the main components of a state model that characterizes human performance, and allows designers and trainers to comprehensively address the most important aspects of error-tolerant design. Furthermore, these performance stages can be modeled computationally, to varying degrees, using standard information processing architectures. This work also demonstrates the effectiveness of a technique using GOMS models to design systems to prevent human error. The technique is applied to WebStock, a realistic web application designed to elicit human error, and the results are used to redesign WebStock's user interface. We compared user performance on the original WebStock interface with the interface improved using the technique. Improvements were made at two levels. *Procedural changes* were those that were directly indicated by GOMS analysis, such as reducing working memory load and optimizing non-intuitive procedures. *Non-procedural changes* were those requiring more analyst expertise but where a GOMS model was instrumental in pointing them out, such as improving the salience of visual objects used in the model. The results showed substantial improvement in task completion time and overall errors, but the GOMS-based procedural improvements were especially important in reducing certain classes of errors. The paper concludes with practical implications of the recovery framework for system and training design such as techniques for supporting error recovery. Further development of the described human-error models will help us to better understand how people commit and recover from errors, and can lead to more robust computer-based tools, improved effectiveness, and reduced training costs.

AUTHORS

Dr. Scott D. Wood is a Senior Scientist at Soar Technology, Inc. He has over ten years of research and industry experience in the areas of software development, e-business consulting, cognitive modeling, and human-computer interaction. His doctoral research included extending GOMS (Goals, Operators, Methods, Selection Rules) modeling to allow for human error, developing techniques for predicting where human errors would occur in an interface, and testing those techniques by applying them to web applications. Dr. Wood also has extensive experience developing human-performance models using the EPIC cognitive architecture, optimizing workflows and interface usability through task analysis, and in designing web solutions for e-business applications. He earned a B.S. in Computer Science (1990) from Tulane University, and M.S. (1994) and Ph.D. (2000) degrees in Computer Science and Engineering from the University of Michigan, Ann Arbor.

Dr. David E. Kieras is an Associate Professor in the Electrical Engineering and Computer Science (EECS) Department at the University of Michigan and also holds an appointment in the Department of Psychology. His primary general research field is applied and theoretical cognitive psychology, with specific interests in human-computer interaction, cognitive simulation modeling, human performance, complex human learning, and natural language processing. His research has been supported by ONR, ARPA, NASA, IBM, and NYNEX Science and Technology, and he has a long-standing interest in human-technology interactions involved in military settings. His research approach is to construct computational models for the cognitive processes involved in tasks that have practical importance, validate the models against empirical data, and prepare them for practical application. His current research focuses on developing the theory, techniques, and tools for analyzing and evaluating usability in HCI, and on the EPIC advance cognitive architecture for human performance modeling

MODELING HUMAN ERROR FOR EXPERIMENTATION, TRAINING, AND ERROR-TOLERANT DESIGN

Scott D. Wood
Soar Technology
Ann Arbor, Michigan

David E. Kieras
University of Michigan
Ann Arbor, Michigan

INTRODUCTION

Designing for human error is a major challenge for developers of safety-critical and mission-critical systems. The literature is replete with examples of how simple human error can result in costly and deadly consequences (Casey, 1993; Perrow, 1984; Reason, 1990). In many of these cases, the real culprit can be traced to the design of the systems in which the error occurred. The errors could have been prevented or their consequences mitigated by designing systems that better support human performance. In purely practical terms, designing systems to reduce errors and their consequences can save time, money, and lives. Kirwan (1992a, b) reviews and compares a large number of techniques for Human Error Identification within the Human Reliability Assessment (HRA) process. He reports that although Human Error Identification techniques exist that, collectively, can deal with all relevant human contributions to error, there are shortcomings. He notes that many of the reviewed methods lack comprehensiveness, are dependent on the ability of the analyst, are not derived from psychological models, and do not provide a direct means for documenting design decisions. An alternative approach that overcomes many of these weaknesses is to base the analysis of the error characteristics of a design on cognitive engineering models of human performance.

GOMS in Error Analysis

GOMS (Goals, Operators, Methods, and Selection Rules) is a family of techniques proposed by Card, Moran, and Newell (1983) (CMN) for modeling and describing human task performance. The use of GOMS to improve design has been well documented (see John and Kieras 1996a, b). Such improvements are possible because GOMS is used to create a detailed characterization of the procedures that the user must follow. This reveals details of what the user must learn, do, and remember, during normal task execution. Although GOMS is often characterized as addressing only error-free behavior, it has a long history of being used in the study of errors (Lerch, 1988; Smelcer, 1989; Byrne, 1993).

Using GOMS for error prevention addresses many of the issues raised by Kirwan concerning the atheoretical nature of the standard error design methodologies and the difficulty of applying them correctly. Since cognitive models are derived from psychological theory, they can often focus more directly on human limitations, thus having the potential to better relate error probability to information that designers can use to improve the design to reduce errors. A second issue raised by Kirwan was that extant error analysis techniques do not provide a good means for documenting design decisions. One of the major benefits of computational cognitive models, is that in order to build a running model, the designer must make the design decisions explicit. Thus, executable models represent a persistent form of design knowledge.

Purpose of this Work

This work shows that a GOMS analysis can also be used to identify error-prone areas of a design and thus guide improvements in the design to sharply reduce the frequency of errors. GOMS was used to identify a variety of potential causes of user errors, in addition to memory errors as documented in the previous research. Furthermore, this work demonstrates how GOMS could be used in a design setting to revise a design to produce fewer errors, and the experimental results confirm that the revised design did indeed produce fewer errors, and in ways expected by the GOMS analysis. The remainder of this paper will first describe a general framework for error recovery. Next, it will summarize the techniques for using GOMS to identify potential causes of user error. Then, the computer application that served as an experimental task domain will be presented with the GOMS techniques used to produce an improved design. The results from an experiment comparing the two designs (see Wood, 2000 for details) will then be presented. We conclude with describing how cognitive engineering models can be used for error-tolerant design and how this work can have an impact on modeling, experimentation, design, and training.

Finally, an important factor that has plagued prior error research has been the difficulty of experimentation on errors – generally it is difficult to devise tasks that are realistic and plausible and at the same time produce errors at a high enough frequency that their causes and patterns can be successfully studied experimentally. This paper presents a realistic experimental domain that successfully produced a significant number of errors, showing how they were affected by systematic design changes. Thus a methodological result of this work is a task domain and approach to the study of errors in HCI that may be useful to other researchers.

A GENERAL FRAMEWORK FOR ERROR RECOVERY

CMN point out that error extensions to GOMS are necessary to fully model even moderately complex

tasks. In their observations of experimental subjects performing a typing task, they noted several interesting results regarding human error. The first result was that errors occurred in 36% of the experimental tasks. This indicates the pervasiveness of human error and the importance of designing to accommodate it. The second result was that the occurrence of an error in a task doubled the average task time. The errors and their correction accounted for an average of 26% of total task time. For one subject, error time accounted for 50% of the time to complete the tasks. Moreover, if an error required real problem solving to correct (e.g. finding one's place in a large text file), task time was increased by an order of magnitude. These results tell us that recovery methods need to be efficient and that they need to be designed such that their use does not require problem solving. A third result was that subjects tended to follow a common path during error recovery. CMN noted that when subjects committed errors, they seemed to progress through five distinct error stages as described in the following excerpt:

1) *Error.* The user makes a mistake.
2) *Detection*. He becomes aware of the error.
3) *Reset*. He resets the editor to allow correction.
4) *Correction*. He undoes the effects of the error.
5) *Resumption*. He resumes error-free activity. (p. 177)

Although these stages were adequate to describe the behavior observed by CMN, several aspects limit their general applicability to other domains. For instance, accurate error-detection and identification was assumed to occur after an error was committed. Sellen and Norman (1992) point out that error identification is not always easy or obvious for users. They recommend that designers and modelers consider error identification separately from detection to better focus on how interfaces can support detection and identification.

A further weakness in the CMN stages is that their Reset stage may not always be possible or necessary. It assumes that users can easily back up, both mentally and within a task, to enable error correction. This is not always possible or desired. Despite these weaknesses, CMN describes a useful approach to error modeling that is structured around specific user mental-stages.

To help generalize the CMN approach, we adopt the following modified structure of user error states:

1. *Error.* The user makes a mistake.
2. *Detection*. The user is aware an error has occurred.
3. *Identification*. The user identifies the error's type.
4. *Correction*. The user corrects the effects of the error.
5. *Resumption*. The user resumes normal tasks.

We refer to this structure of states collectively as error recovery stages. This differs from the CMN structure in two fundamental ways. First their Reset stage has been removed and is considered an implicit part of the error correction stage, if a reset is used at all. Second, error identification is split from detection to allow better focus on the identification process.

An important characteristic of the recovery stages is that the user's progression through them is not necessarily a linear process. How and why users move from one stage to another is critical to understanding error recovery. To clarify this process, we can view the user's progression through the recovery stages as movement between a set of mental states.

Figure 1 illustrates how the recovery stages fit into a simple state diagram of the user's mental-states during error recovery. From the user's perspective, the error recovery model is straightforward. During normal, routine performance, the human does everything right and continuously executes correct actions. I refer to this as the *Normal* state. But, when an error occurs, the human enters a Quasi-Normal state where everything seems normal, but where some failure is imminent. The transition between the *Normal* and *Quasi-Normal* states reflects the Error stage. The user can continue performing correct actions within the *Quasi-Normal* state until the user detects that something is wrong, prompting him or her to recover. This transition from the *Quasi-Normal* state to the *Recovery* state reflects the Detection stage. Once in *Recovery*, the user identifies the error (the Identification stage) and takes the necessary corrective actions (the Correction stage). When error correction is complete, the user returns to normal operations (the Resumption stage).

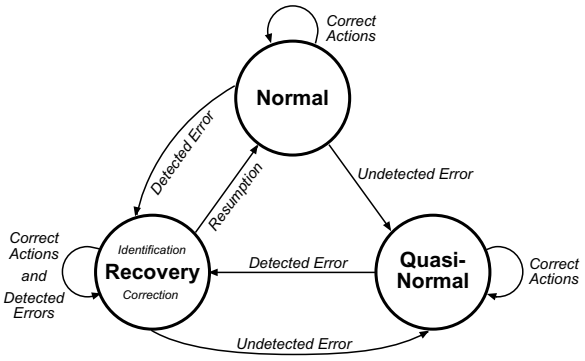


Figure 1. A general framework for error recovery. The state diagram illustrates user mental states while moving through error recovery stages.

Two additional transitions reflect the nonlinear nature of error recovery. The first of these occurs when the user detects an error as the action is performed (as with the CMN errors). In these cases, the user jumps immediately to the Recovery state. This transition can occur from any state, including Recovery. The second transition type can occur when an undetected error occurs during the Recovery state. Here, the user reenters a Quasi-Normal state, where error correction seems to be proceeding, but where another failure is imminent.

Movement through the mental states can be

illustrated with a simple example: The task of entering your office in the morning. A possible procedure for this is represented in Listing 1. The On_error syntax indicates the name of a correction procedure, Remove key and restart current goal, with the normal procedure, Enter office. If any of the steps in the normal procedure fail, control would be transferred to the correction procedure. The task steps are: find the correct key, insert it into the lock and turn it, verify that the door is unlocked, and open the door and step through. If, during Step 1, you fail to find the right key and instead locate the wrong key, you would move from a Normal State to a Quasi-Normal State. An error had occurred in Step 1, but it was not yet detected. In Step 2 the key may not fit or may not turn, so Step 2 would fail, indicating that something was wrong. You would then move to a Recovery State and start the procedure of removing the key and trying again. The correction method includes the transition from the Recovery State back to the Normal State.

```
Method_for_goal: Enter office
On_error: Remove key and restart current goal.
Step 1. Look for and choose correct key.
Step 2. Insert and turn key.
Step 3. Verify door is unlocked else go back to
step 1.
Step 4. Open door and step through.
```

Listing 1. GOMS excerpt showing potential memory overload error.

Alternatively, if Step 1 had been successful, but you turned the correct key incorrectly (like in the wrong direction), the error would cause a transition from the Normal State to the Quasi-Normal State. Here, the error might only be detected when you attempted to verify that the doorknob turns. The explicit check in Step 3 would indicate a problem, which would cause the transition to the Recovery State. Here, the recovery plan specifies that you go back to Step 1 and resume normal task execution. So to recover, you simply to make the transition from the Recovery State back to the Normal State and try again.

The Framework is important for interface designers because it makes clear several important issues when designing for error tolerance:

1. There may be a delay between when an error occurs and when it is detected. Any actions made during the Quasi-Normal stage may need to be undone anyway, so efforts should be made to help users detect errors as soon as possible after the errors are made. Furthermore, because there is a potential delay, it may be difficult for the user to identify the source of the error and choose an appropriate corrective procedure. Interface aspects that help identify errors and their corrective actions will reduce the time users need to spend in real problem-solving to correct problems.
2. Errors can occur while in a Recovery State, so error correction methods need to have their own recovery methods.
3. Correcting an error is not the end of recovery. We also need to consider how users will resume normal task performance; system design should reflect this.

USING GOMS TO IDENTIFY ERROR SOURCES

As proposed by Wood (1999), the basic approach for using GOMS to design for error tolerance is to first construct a GOMS model for the task and then to examine both static and dynamic aspects of the model to identify sources of error. These sources can be classified as either procedural or non-procedural aspects of the design. Procedural aspects concern the effects of the action sequences in the methods, while non-procedural aspects concern the perceptual and conceptual qualities of the objects in the environment. For example, consider the GOMS methods sketched in Listing 2 for selling a stock in a notional stock management application. The procedural aspects involve the steps that the user must perform to sell a stock, namely, looking for and remembering number of shares and stock symbol (e.g. MSFT) information, then performing a selling procedure. The methods require finding and remembering two values, the stock's symbol and the number of shares to be sold, and then going to a second page where this information must be typed in; this pattern indicates a possible memory load problem. In addition, the procedures are relatively slow to execute because the application has not pre-positioned the cursor in the first entry field.

```
Method for goal: Sell stock
Step 1. Look_for symbol and store under
<symbol_to_sell>.
Step 2. Look_for shares and store under
<shares_to_sell>.
Step 3. Accomplish goal: Sell stock using
<shares_to_sell> and <symbol_to_sell>.

Method for goal: Sell stock using <shares> and
<symbol>.
Step 1. Press "Sell" button.
Step 2. Wait for page to load.
Step 3. Look_for "Shares" field.
Step 4. Point_to "Shares" field.
Step 5. Type_in <shares>.
Step 6. Look_for "Symbol" field.
```

Listing 2. GOMS excerpt showing potential memory overload error.

To illustrate the identification of non-procedural aspects, notice that by inspection of the methods, one can tell that on one page the user must visually search for the stock symbol, and on another page must find the proper field to type the information into. Thus the form, typography, and layout of the symbol text and field label on the screen will affect the time and reliability of the "Look_for" operators in the methods. In contrast to procedural changes, correcting any problems with these aspects of the interface does not change the structure of the user's procedures, but will affect how quickly and accurately they can be executed. Also note that the model identifies the essential information for this task. Thus the model is a straightforward way to identify critical visual aspects for relevant interface elements.

A static analysis of a GOMS model considers just the content of methods in the model, such as that shown in Listing 2. The above examples of the appearance of the symbol information and field, and the lack of cursor-prepositioning can be deduced by such a static inspection, in which one simply examines the GOMS methods for certain patterns of steps and operators (see Wood, 1999). A dynamic analysis involves running the model as a simulation to execute a set of specific task instances in which all necessary information such as stock symbol and shares are specified. For example, the memory load during the execution of a method may be very high in one task instance (e.g. if multiple stocks must be compared and traded), but low in another. Likewise, some tasks may result in different patterns of visual search, some of which might be more error prone than others, depending on the non-procedural aspects of the information involved.

Using GOMS to design for error prevention yields three main benefits. First, GOMS models can address procedural sources of error by identifying locations, times, and conditions within a task where errors are likely. Second, GOMS models can be used as a road map to guide static and dynamic analyses of non-procedural sources of error. Third, since GOMS models are psychologically motivated and can provide specific information for design improvement, they address many of the criticisms discussed by Kirwan (1992a). To help demonstrate these claims further, these analysis techniques were applied to the redesign of a web application and tested for their effectiveness at reducing human error.

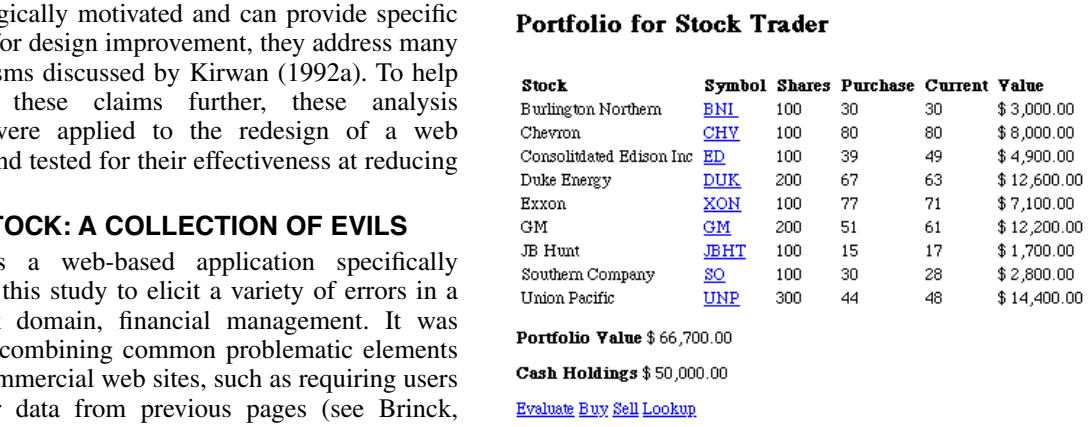
WEBSTOCK: A COLLECTION OF EVILS

WebStock is a web-based application specifically designed for this study to elicit a variety of errors in a realistic task domain, financial management. It was designed by combining common problematic elements of several commercial web sites, such as requiring users to remember data from previous pages (see Brinck, Gergle, and Wood, 2001, for examples). WebStock includes functionality for basic portfolio management including buying, selling, and evaluating stocks, and looking up stock symbols. Generally, the user will login to the system and perform a series of portfolio management tasks. Tasks consist of evaluating and comparing stocks and buying or selling them. If the stock symbol (e.g. MSFT) is not known, the user has to look it up prior to evaluation. To evaluate a stock, WebStock provides the user with a fictional Evaluation number for each stock. The user must determine where the Evaluation number falls within a range to decide whether to buy, sell, or hold a stock. The tasks users performed were of just two types, Buy and Sell, but these required various subtasks to be performed along the way. An example task is "Evaluate the Airline candidates and buy 100 shares of the one with the strongest buy recommendation, if appropriate."

The Original Interface

To illustrate the usability problems in the Original interface, Figure 2 shows the Portfolio page, the main page of the WebStock application from which the tasks are started. This page contains an overview of the user's stock portfolio, providing names, stock symbols, the number of shares owned, and the current value of the portfolio. The links at the bottom allow the user to evaluate a stock, buy a stock, sell a stock, and look up a stock symbol. Perceptual quality was poor because all pages had a medium gray background color.

The labels "Evaluate" and "Lookup" are somewhat ambiguous and the left-to-right ordering is backwards because the evaluation function requires the stock symbol, which is obtained by using the Lookup function. The portfolio page is not automatically updated after a transaction – the user is expected to refresh the browser cache with a reload action. Other usability problems include a mismatch between window title and page title for all sub-pages, and information that is extraneous to the tasks (e.g. the purchase price requires time to read and can be confused with other information on the screen, even though it is not required by the tasks)



Portfolio for Stock Trader

Stock	Symbol	Shares	Purchase	Current	Value
Burlington Northern	BNI	100	30	30	$ 3,000.00
Chevron	CHV	100	80	80	$ 8,000.00
Consolidated Edison Inc	ED	100	39	49	$ 4,900.00
Duke Energy	DUK	200	67	63	$ 12,600.00
Exxon	XON	100	77	71	$ 7,100.00
GM	GM	200	51	61	$ 12,200.00
JB Hunt	JBHT	100	15	17	$ 1,700.00
Southern Company	SO	100	30	28	$ 2,800.00
Union Pacific	UNP	300	44	48	$ 14,400.00

Portfolio Value $ 66,700.00

Cash Holdings $ 50,000.00

Evaluate Buy Sell Lookup

Figure 2. The main WebStock screen is the Portfolio page. Here, users can view their current stock holdings and perform other tasks using the navigation links.

Redesigning WebStock

A GOMS model of WebStock was written in the GOMSL (GOMS Language) notation and run using EGLEAN (Error-extended GOMS Language Evaluation and ANalysis), a variant of GLEAN (Kieras, et al., 1995, Kieras, 1998), which was built to study human error recovery (Wood, 2000). The model was used systematically to guide the redesign along the dimensions already described. Here we present details and examples of how the techniques were applied.

Example Procedural Improvements from a Static Analysis. The static analysis of procedural error sources consisted of locating patterns in the GOMSL model suspected of inducing errors, such as post-completion, mode, capture, and calculation errors

(Wood, 1999). For example, post-completion errors were identified in the GOMSL model by looking for patterns in which there are cues accompanying the completion of a goal, but where there are additional steps in the method that must be subsequently executed. These additional steps are likely to be omitted. Listing 3 shows an excerpt from the Buy method in WebStock that illustrates an example of this pattern. The steps not shown identify the stock to buy, and then Step 13 invokes a method to accomplish the goal of actually buying the specified stock; this method involves responding to a confirmation page to complete the purchase transaction, which is the cue that the goal has been accomplished. However, after navigating back to the Portfolio page, the user must remember to press the Reload button to refresh the page. Some of these post-completion steps should be prone to being omitted; if so, the user will commit various later errors if the task calls for making use of the out-of-date information. The error can be prevented by altering the application so that it always updates the page or does not allow it to be cached by the browser.

Examples of other procedural changes based on static analysis included reducing Mode errors involved in selecting type-in fields (along the lines described above) and improving error recovery routines.

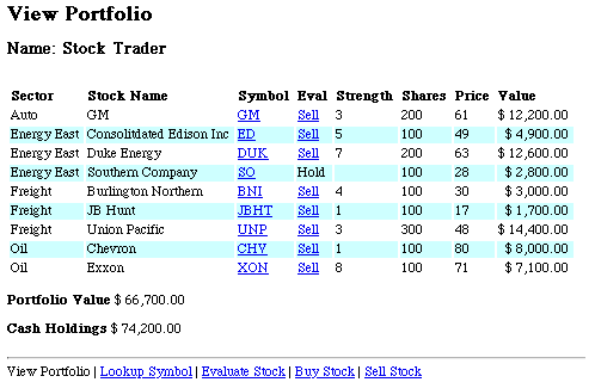
```
Method_for_goal: Evaluate_and_buy_best_candidate_of
Sector using <sector>, and <shares>
…
Step 13. Accomplish_Goal: Buy Stock using
   <best_symbol>, and <shares>.
Step 14. Accomplish_Goal: Navigate_to Portfolio.
Step 15. Accomplish_Goal: Press button using "Reload".
Step 16. Wait_for_visual_object_whose
   type is "gestalt",
   and status is "finished_loading"
   and_store_under <load_status>.
Step 17. Return_with_goal_accomplished.
```

Listing 3. GOMSL excerpt showing potential Post-completion error. The "gestalt" item in Step 16 represents the overall appearance of the page.

Example Procedural Improvements from a Dynamic Analysis. The main dynamic analysis of procedural error sources consisted of simulating working memory usage of users performing the WebStock experimental tasks. Execution trace files generated by EGLEAN for the experimental tasks were run through a Perl script to collect statistics on the peak working memory load during the execution of each method and the peak time that working memory items were kept in memory. A threshold based on the averages of these was set and used to locate methods that produced high memory loads, and items that were kept in memory for a long time. The methods and items were then examined, and the interface modified to reduce the memory requirements. For example, on the Portfolio page (Figure 3), certain items of information are needed for the Sell tasks; in the Improved interface these items are all present, whereas in Original interface they had to be looked up and remembered. Not only does this reduce the need to remember critical information, but fewer steps are now required in the Sell tasks.

Non-Procedural Improvements. Both static and dynamic analyses of non-procedural error sources were conducted using the GOMS model as a guide. For example, an improvement based on a static analysis included modifying the portfolio page with alternating background colors to reduce confusions between rows.

Other non-procedural improvements were based on a dynamic analysis. For example, the text used for navigation links was made less ambiguous by basing it on the actual Goals in the GOMS model. Here, *Lookup* and *Evaluate* were changed to *Lookup Symbol* and *Evaluate Stock*. Finally, using dynamic information, the links were ordered according to their typical usage in a buy task.



View Portfolio

Name: Stock Trader

Sector	Stock Name	Symbol	Eval	Strength	Shares	Price	Value
Auto	GM	GM	Sell	3	200	61	$ 12,200.00
Energy East	Consolidated Edison Inc	ED	Sell	5	100	49	$ 4,900.00
Energy East	Duke Energy	DUK	Sell	7	200	63	$ 12,600.00
Energy East	Southern Company	SO	Hold		100	28	$ 2,800.00
Freight	Burlington Northern	BNI	Sell	4	100	30	$ 3,000.00
Freight	JB Hunt	JBHT	Sell	1	100	17	$ 1,700.00
Freight	Union Pacific	UNP	Sell	3	300	48	$ 14,400.00
Oil	Chevron	CHV	Sell	1	100	80	$ 8,000.00
Oil	Exxon	XON	Sell	8	100	71	$ 7,100.00

Portfolio Value $ 66,700.00

Cash Holdings $ 74,200.00

View Portfolio | Lookup Symbol | Evaluate Stock | Buy Stock | Sell Stock

Figure 3. The improved WebStock portfolio screen.

Differential Improvements

It can be argued that the non-procedural problems could have been identified without a GOMS model by just applying ordinary usability guidelines. Hence, simply showing that the redesigned interface is better than the original would not strongly support the utility of GOMS in error-tolerant design. Therefore, it is important to show that using a GOMS model to guide the redesign produces distinctive benefits. The changes most motivated by GOMS are the procedural changes, and the way procedural changes would most affect errors would be in reducing memory load. Thus, the non-procedural changes were applied uniformly across the entire WebStock interface, improving both the Buy and the Sell tasks, but procedural changes were also applied almost exclusively to the portions of the interface involved in Sell tasks. The prediction was that errors resulting from procedural problems would mainly show up as memory errors, and these would be reduced for the Sell tasks, but not for Buy tasks, while the other error types would be reduced uniformly across both task types. In addition, the time required to complete the tasks should be improved more for the Sell tasks than for the Buy tasks.

The users of both interfaces performed almost the same actions for the Buy tasks. These tasks required participants to first navigate to a Lookup table to look up the stock symbol for each stock candidate. They had to remember the stock symbol, navigate to the Evaluation page, type in the symbol, and find the evaluation information on the Stock Detail page. They then compared each candidate stock using the evaluation information, navigated to the Buy page, typed in the stock symbol and desired shares, and pressed the "Buy" button. This brought up the Confirmation page. Participants using the Original interface navigated back to the Portfolio using the "Back" button and pressed the "Reload" button. Participants using the Improved interface instead navigated back to the Portfolio using the direct link from the Confirmation page and the Portfolio page was reloaded automatically.

The Sell tasks were very different for the two interfaces. In the Original interface, the Sell tasks required the participants to navigate to each stock's Detail page, get its evaluation value, determine whether it was in the required sell range and compare it to any other stocks in the stated category for that task. They then had to remember the stock's symbol and the number of shares owned, and navigate to the Sell page. After entering the Selling information in the designated entry fields, the participants pressed the Sell button, confirmed the transaction, navigated back to the Portfolio page, and pressed the Reload button.

In the Improved WebStock interface, all of the Detail information necessary for the Sell tasks was displayed on the Portfolio page. The participants had to compare the evaluation information for each stock being considered, pick the one with the strongest evaluation strength, and press a "Sell" link to go to the Sell page. The stock symbol and shares were automatically entered into the Sell form as default values, so the participants then simply confirmed the information and pressed the "Sell Stock" button. They confirmed the transaction on the Confirmation page and pressed a link to go back to the Portfolio page. The Portfolio was automatically reloaded, so they did not have to reload it manually.

EXPERIMENT

The experiment was a simple comparison of the Original and Improved versions of the WebStock system. A separate group of participants used each system to perform a series of tasks and their actions were recorded and analyzed for time required and errors. In keeping with the realistic task domain, the participants were not heavily practiced with the system, but did have an opportunity to learn the system before performing the tasks whose data are presented here. The participants were asked to work as quickly as possible, but to "think aloud" while performing the tasks so that their intentions could be compared to actual actions.

Method

Procedure. The experimental tasks were to buy and sell a list of stocks using WebStock, using either the Original WebStock interface or the Improved WebStock interface. The participants were instructed to act as stock traders, buying and selling stocks specified in general terms by their managers, using a supplied list of candidate companies, grouped by category, to consider. Eight tasks were performed. The first was a very simple Login task, whose results are not included in this paper; the remainder consisted of four Sell tasks interleaved with three Buy tasks. The first Sell and Buy tasks were very simple, intended to allow the participants to learn how to use the system; their results are likewise not included in this paper.

Participants were instructed to complete the list of tasks in order and to return to the main portfolio page after each task. They were reminded to reload the portfolio page if necessary to update its information after each transaction. They were also told that if they made a mistake or if the system was "buggy," that they should attempt to correct the problem as quickly as possible and continue. They were instructed to think-aloud as they worked.

It is important to note that subjects were not given any direct training and very little instruction on how to use the WebStock program; rather, like most web applications, they had to learn how to use the program by interacting with it, following the instructions or cues presented in the web pages. Similarly, they received no feedback during the experiment on whether they performed the tasks correctly.

Participants and Design. Participants were university students and staff, all of whom were familiar with using web browsers and spent at least 1 hour per week browsing the web. The data were collected first for 11 participants using the Original interface, then for 12 participants using the Improved interface. The two WebStock interfaces, the Original and the Improved, were a between-subjects factor; the tasks (and task type) were a within-subjects factor.

Apparatus. The experiment was run using one computer to serve the web-application and another to host the client browser used by the participants. Participants interacted with WebStock using Netscape Communicator 4.51 running on an Apple PowerMac 9500 computer using a 17-inch monitor at a resolution of 832 x 624 pixels. Client and server machines were connected on a 100 Mbit/sec Ethernet network. WebStock pages included JavaScript code to collect timestamped keystroke and mouse events. Participant's speech and their screen actions were videotaped.

Results

The complete data set and its analyses are complex, so only the most central subset of the results are presented here; see Wood (2000) for more detail. An error was defined in the context of the experimental tasks. First, the optimal action sequence specified by a GOMS model for performing the experimental tasks was determined. Then each participant's action sequence was compared with the GOMS optimal sequence; any

.
Table 1. Observed errors by category for each interface.

Error Categories	Interface					
	Original			Improved		
	Number observed	Errors / subject	Percent of all errors	Number observed	Errors / subject	Percent of all errors
Timing	11	1.00	0.069	3	0.26	0.050
Action/object	22	2.00	0.138	7	0.58	0.117
Motor	24	2.18	0.151	10	0.83	0.167
Perceptual	14	1.27	0.088	9	0.75	0.150
Memory	46	4.18	0.289	23	1.92	0.383
Omitted steps	14	1.27	0.088	4	0.33	0.067
Post-completion	15	1.36	0.094	4	0.33	0.067
Typing Field	13	1.18	0.082	0	0.00	0.000
Total	159	14.45	1.000	60	5.00	1.000

deviations were considered possible errors. Whether they were errors in the context of the task was then determined by consideration of what method was under way at the time, by reference to notes kept during the experiment, and additional actions and think-aloud utterances found on the videotapes. Inefficient actions, such as exploring the interface, were not considered errors; many such actions were observed during the initial Buy and Sell tasks as the participants were learning how to use WebStock. Likewise, alternative procedures that still allowed the task to be completed successfully were not treated as errors. For example, some participants chose to navigate back to the Portfolio page by pressing the Home button (taking them back to the Login screen) and then re-executing the Login procedure (taking them to the Portfolio page). Each error was assigned to one of a set of 23 contextual error types, which were grouped into a set of 8 general categories (see Wood, 2000, for details).

To illustrate how these contextual errors were classified, in one case the participant evaluated three stocks correctly by navigating to the respective Detail pages, but then departed from the optimum sequence by going back to a Detail page and re-evaluating one of the stocks. Since the only information that the user could get from the Detail page is the evaluation value, this pattern of going back to a Detail page is a strong indicator that the participant forgot the evaluation value. This conclusion was confirmed by observation notes and videotape that the participant said something indicating uncertainty about the value; thus this event was classified as a Forgot Evaluation error. In some cases, exact classification of a possible error was not possible due to multiple possible causes. For example, if a participant sold the wrong stock, it could have been because the participant erred in the evaluation process (Calculation Error), or confused two stock symbols (Memory Corruption), or because the participant misunderstood the instructions for the task. In such ambiguous cases, errors were classified simply as Erroneous Actions.

Observed Errors. The observed errors are presented in Table 1. Shown for each interface and general category are the number of observed errors, the number of errors per subject (to compensate for the unequal numbers of participants), and the proportion of the total for each interface. A key point is that although the experiment involved a modest number of subjects and only a few tasks, a substantial number of errors were observed, demonstrating a successful approach to studying errors in a realistic task setting. There are clearly substantially fewer errors in the Improved interface compared to the Original, showing that using the GOMS technique for error reduction appears to be successful. In particular, note how some types of error, such as Typing Field errors, were completely eliminated in the Improved design, showing that some forms of error can be completely designed out of an interface. In addition, note that some error types, such as Memory errors in the Buy tasks for Improved interface, were intentionally not reduced, so the improvement is relatively less compared to other error types. Also note that because some error types, like Typing Field errors, were sharply reduced, other error types increased in relative frequency.

As discussed earlier, differential improvements were made to the Original interface to see if procedural improvements, which are distinctively based on GOMS, had distinct effects compared to non-procedural improvements, which in this case were based on a GOMS analysis, but could have been identified by other means. The Improved interface had only non-procedural improvements for Buy tasks, and both procedural and non-procedural improvements for the Sell tasks. The most clear-cut measure of the value of the procedural improvements is a reduction in task performance time (due to shorter, more efficient, measures), and more relevantly, a reduction in errors produced by memory failures – the Improved interface procedures required less information to be kept in working memory for shorter periods of time compared to the Original interface. Demonstrating these effects was done by comparing the two interfaces in terms of the two types of task, the Buy and Sell Tasks, and the number of Memory versus Non-Memory errors.

For simplicity in analyzing the data, the groups were

made equal in size by dropping one of the subjects from the Improved interface group; the deleted subject was one of two who tied for the smallest number of errors, and was the one with the smallest number of errors per action. Note that this choice was conservative in that it would be expected to reduce the effect of the improvements in the interface. The Login task and first Buy and Sell tasks were not included, because on these, the participant would be learning how to use WebStock, meaning that their errors would be of a different nature than after having some familiarity with the program. The remaining time and error data was then aggregated into the Buy and Sell task types, and then subjected to an analysis of variance with 11 subjects per group, with the factors being Interface (Original vs. Improved), Task type (Buy vs. Sell), and for the error data, Kind of error (Non-memory vs. Memory). Unless otherwise stated, all main effects and interactions claimed here were significant at the 0.05 level or better.

Table 2 summarizes the results for the time required to complete each task and the number of errors observed per subject per task. Compared to the Original interface, the Improved interface resulted in substantially faster task performance, a 32% improvement overall, but the effect was especially strong for the Sell tasks (51%) compared to the Buy tasks (18%), reflecting the additional benefits of the procedural changes. A similar pattern appears for the mean number of errors for Buy and Sell tasks. The Improved interface produced 58% fewer errors than the Original, but the effect for the Buy tasks was only 34%, while it was huge (83%) for the Sell tasks, again demonstrating the substantial benefits of the procedural improvements.

Table 2 includes a finer breakout of the number of errors into Non-Memory errors and Memory errors. Note how in Buy Tasks, the Non-memory errors are considerably reduced by the Improved interface, but there is no improvement (actually a slight decrement) in Memory errors. However, for Sell Tasks the Improved interface reduces both kinds of errors substantially and almost equally. Despite the clarity of this effect, demonstrated by the corresponding pattern of significance in simple t-tests, the corresponding three-way interaction is not at all statistically reliable. As an alternative that represented the mathematical structure of the data better, a log-linear analysis was conducted. The analysis used the error frequencies for all of the subjects, totaled for the same Buy and Sell task types, and aggregated over Subjects (which is known to yield a conservative analysis). The only model that fit the data well was the saturated hierarchical model that includes the three-way interaction; that is, the frequency of errors depends significantly on the combination of Interface, Task type, and Kind of error. This confirms that the Non-Memory error rate was improved by the Interface for both Task types, but the Memory error rate was improved substantially only in the Sell tasks, where the procedural changes were made to reduce the memory load. This result shows that the long-recognized ability of GOMS to aid in designing good procedures for interfaces is also directly relevant to reducing at least one important class of errors. However, the non-procedural errors were also substantially reduced in the Improved interface; these improvements were made by systematically applying a GOMS analysis to help identify potential sources of error.

Table 2. Experiment results summary.

	Interface		
	Original	Improved	% Improved
Time (seconds)			
Buy Tasks	193	158	18%
Sell Tasks	142	70	51%
Mean	167	113	32%
Errors			
Buy Tasks	1.41	0.93	34%
Sell Tasks	1.44	0.24	83%
Mean	1.42	0.59	58%
Errors by Kind			
Buy Tasks			
Non-Memory	2.09	1.09	48%
Memory	0.73	0.77	-5%
Sell Tasks			
Non-Memory	1.97	0.39	80%
Memory	0.91	0.09	90%

USING COGNITIVE ENGINEERING MODELS FOR ERROR TOLERANT DESIGN

A central theme in designing for human error is to build a multi-layered defense (c.f. Reason, 1996). Designing for human error effectively requires addressing several aspects of error management, including prevention, reduction, detection, identification, recovery, and mitigation. Cognitive models can be used effectively in each of these areas to create a multi-layered defense.

Prevention. Eliminate the potential for error to occur by changing key features of the task or interface. Cognitive modeling can be used to help identify the root causes of human error and to detect error-inducing patterns that other approaches do not.

Reduction. Reduce the likelihood that the user will get into an error state when prevention is not possible by ensuring the user is aware of action consequences and by training on both normal and recovery procedures. A by-product of cognitive modeling is a complete set of operating procedures, the foundation for task-based training materials.

Detection and Identification. Ensure that if the user does err, the system makes it easy for the user to detect and identify the error. Cognitive modeling can be used to determine where and how a user is most likely to detect and identify errors, allowing designers to make informed decisions about diagnostic support.

Recovery. Once an error has been detected and

identified, the system should facilitate rapid correction, task resumption, and movement to a stable system state. A modeling approach can determine if recovery methods are achievable, complete, and comprehensive.

Mitigation. Minimize the damage or consequences of errors if they cannot be recovered from. Even when all other error management steps have been taken, errors will still be made, so systems should be designed such that catastrophic outcomes from human error are not possible. Cognitive modeling can indicate where such errors are most likely and help determine where designers need or need not focus their mitigation efforts.

IMPLICATIONS

This work has both theoretical and practical implications for many aspects of both military and commercial use of complex systems in the areas of modeling, experimentation, system design, and training.

Modeling

The Framework can drive extensions to GOMS and other engineering models by specifying the infrastructure needed to model erroneous behavior. Wood (2000) provides a comprehensive description of the mechanisms and GOMS extensions necessary to model all aspects of human error in a design context. For example, the Framework indicates that there must be a means for the GOMS cognitive processor to detect errors, and that there must be control mechanisms to move to and from correction routines. It indicates that the source of an error and how it is expressed are not necessarily the same. It also demonstrates why hierarchies of recovery are necessary because of errors that can occur during the recovery state.

Experimentation

The Framework can likewise facilitate laboratory research on human error by suggesting locations for expected errors and error recovery. One of the techniques used to design the WebStock experiment was to map potential error types onto the Framework to determine interesting manipulations and ascertain what erroneous behavior to look for. For instance, one similarity between the Buy and Sell tasks in WebStock was in the appearance of the Buy and Sell screens. They both had fields for stock symbol and shares, and buttons for submitting the transaction and clearing the fields. To set up the situation for a Mode error, the links for each page were placed next to each other. This increased the likelihood that participants would click on one link when the other was intended. Then, using the Framework, all but one cue indicating which page the participant was on were removed to reduce the likelihood of detecting the mode error. Detection was made more difficult by making the subsequent transaction confirmation page identical for the Buy and Sell transactions.

System Design

The Error Recover Framework guides error-tolerant design by serving as a road map for design efforts. For example, of all the transitions in the Framework, the only one that does not involve some amount of wasted effort is the Correct Actions transition in the Normal state. If the user is in either the Quasi-Normal or Recovery state, then an error has occurred and there is an associated cost for getting back to normal activities. The obvious message here is to prevent as many errors as possible. However, the Framework also shows why early error detection is important. The longer the user spends in the Quasi-Normal state, the more wasted actions. Finally, because actions outside of the Normal state are useless to normal task performance, the Framework demonstrates the importance of efficient error recovery procedures that return the user to the Normal state rapidly.

The Webstock experiment further demonstrates the utility of GOMS for many real-world design tasks because it is at a necessary granularity for most design. Many software tasks where error-tolerance is important can be viewed as hierarchical sequences of discrete keystroke-level tasks. Even if tasks are combined or used in a non-hierarchical manner, the hierarchical-sequential view represents a lower bound on task performance. That is, regardless of the conditions under which people are performing a task, they cannot exceed the performance specified in an ideal, hierarchical model. Any performance gains designers hope to provide in normal operating conditions must first be achieved under ideal conditions. The same argument applies to error management. If we do not achieve acceptable usability for the simple case of hierarchical, sequential performance, we cannot achieve acceptable performance under real-world conditions. Furthermore, the same qualities of GOMS analysis that make it useful for error-free design issues also make it useful for error-tolerant design. GOMS is simple enough to apply relatively quickly, and contains enough detail to point designers to fundamental problems.

Training

Usability, trainability, and error-tolerance are inextricably linked. Error-tolerance is an important component of usability and the same error-tolerant qualities that make a system highly usable are the same that will also make it highly learnable. One well-established empirical finding in the human-computer interaction literature is that systems that are easy and fast to use are also easy to learn. This finding conflicts with conventional wisdom, but actually follows directly from the theoretical analysis contributed by the engineering models. In general, in order to learn to use a system, the person must learn the procedural knowledge represented in the GOMS model, together with any additional declarative (factual) knowledge required to perform the task (such a definitions of symbology, vocabulary, or concepts used in the interface information). If the design of the system is such that the required procedural knowledge has been made as simple

and consistent as possible, and the need for additional declarative knowledge has been minimized, then the system will be easy to learn. However, these same factors tend to make the system easy and fast to use as well: Methods that are short, simple, and consistent tend to be fast to execute compared to convoluted and inconsistent methods. If the interface requires only a minimum amount of extra declarative knowledge, then there will be fewer pauses while trying to remember some bit of knowledge, and fewer memory failures that cause errors whose correction and recovery demand additional learning and execution time. In short, by designing for trainability we will also be improving usability, and by designing for usability, we can also improve trainability.

CONCLUSIONS

Human error is pervasive. Designing for human error should also be pervasive. Military and commercial entities are depending on, and in some cases, betting everything on, increasingly complex computer systems to provide a competitive edge and increased productivity. Yet, without usable, error-tolerant interfaces these enterprises have a much greater risk of failure. Moreover, many large-scale software development efforts do not have the budget to do extensive error analysis. Software developers need simple, practical techniques to guide the design of systems that will comprehensively manage human error. This work also contributes to the software and user-interface development community by illustrating some of the perils of poor error-tolerance in the web domain and providing a technique for reducing errors and their associated risks. But prevention alone is not enough. The Framework for Error Recovery can assist in the design of comprehensive error management and help guide human error modeling at an engineering level.

The Webstock experiment demonstrates that human error can be reduced substantially by using a GOMS model both to help identify potential error sources and motivate design improvements to mitigate their effects. The GOMS analysis contributes to reducing error in two ways: First, it contributes directly by identifying procedural problems, such as unnecessarily long or complex procedures, and procedures that impose a high memory load. Second, it contributes indirectly by guiding consideration of the non-procedural sources of error. Compared to ordinary guidelines, this structured approach can help designers to systematically inspect traditional sources of error, consider their importance, and explore their consequences for users. Finally, the experiment provides an experimental domain that may be useful for future error research. Future work should explore and validate these contributions further, and lead to the development of tools that will make it easier for designers to identify and correct error-prone interface and task elements.

REFERENCES

Brinck, T., Gergle, D., & Wood, S.D. (2001). *Usability for the Web.* San Francisco, CA: Morgan Kaufmann.

Byrne, M. D. & Bovair, S. (1997) A working memory model of a common procedural error. *Cognitive Science, 21*, 31-61

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

Casey, S. (1993). *Set Phasers on Stun*. Santa Barbara, CA: Aegean Publishing Co.

John, B. E., & Kieras, D. E. (1996a). Using GOMS for User Interface Design and Evaluation: Which Technique? *ACM Transactions on Computer-Human Interactions, 3*(4), 287 - 319.

John, B. E., & Kieras, D. E. (1996b). The GOMS Family of User Interface Analysis Techniques: Comparison and Contrast. *ACM Transactions on Computer-Human Interactions, 3*(4), 320 - 352.

Kieras, D. E., Wood, S. D., Abotel, K., & Hornof, A. (1995). *GLEAN: A Computer-Based Tool for Rapid GOMS Model Usability Evaluation of User Interface Designs.* Paper presented at the ACM User Interface Software and Technology, Pittsburgh, PA.

Kieras, D.E. (1998). *A guide to GOMS model usability evaluation using GOMSL and GLEAN3*. (Technical Report No. 38, TR-98/ARPA-2). Ann Arbor, University of Michigan, Electrical Engineering and Computer Science Department.

Kirwan, B. (1992a). Human error identification in human reliability assessment. Part 1: Overview of approaches. *Applied Ergonomics, 23*(5), 299-318.

Kirwan, B. (1992b). Human error identification in human reliability assessment. Part 2: Detailed comparison of techniques. *Applied Ergonomics, 23*(6), 371-381.

Lerch, F. J. (1988). *Computerized Financial Planning: Discovering Cognitive Difficulties in Model Building.* Doctoral dissertation, University of Michigan.

Perrow, C. (1984). *Normal Accidents*. New York, NY: Basic Books, Inc.

Reason, J. (1990). *Human Error.* New York: Cambridge University Press.

Sellen, A. J., & Norman, D. A. (1992). The Psychology of Errors. In B. J. Baars (Ed.), *Experimental Slips and Human Error.* New York: Plenum Press.

Smelcer, J. B. (1989). *Understanding User Errors in Database Query.* Doctoral dissertation, The University of Michigan.

Wood, S.D. (2000). Extending GOMS to human error and applying it to error-tolerant design. Doctoral dissertation, University of Michigan.

Wood, S. D. (1999). *The Application of GOMS to Error-Tolerant Design.* Proceedings of the 17th International System Safety Conference, Orlando, FL.