

Managing Your Defense Against GUI's from Hell

— published in *The X Journal*, September/October, 1995



Eric Schaffer, Ph.D., CUA, CPE, is Founder and CEO of Human Factors International, Inc. (HFI). He teaches, consults, and speaks on corporate and governmental Web and GUI interface design issues.



John Sorflaten, Ph.D., CUA, CPE, teaches and consults as a Project Director at HFI. With Eric, he initiated a usability curriculum at a local university in his home town of Fairfield, IA.

©1996, Human Factors International, Inc.

This new series of articles will give you a front-line report from the GUI revolution in corporate North America. Careful reading may save you from some major combat losses in user interface (UI) design. The combat metaphor is real because the enemy is wily and pervasive. We'll give you examples. At our screen design seminars, we pass out warning buttons labeled "GUIs from Hell". We see the enemy as decisions made in the name of "design" but that lack the soul of design. We call the enemy "cryptodesign." Cryptodesign arises when a design that works for certain situations gets used in different, inappropriate situations.

Cryptodesign misleads the unwary Check the number of times you walk out of an office complex grasping a door handle shaped to say "pull me" while warning you with a label that says PUSH. The unwarranted generalization of "handle" to both sides of a one-way door shouts cryptodesign at work. You've see your VCR mercilessly flashing 12:00 pm into the night (and day), reminding you of your slow-witted inability to set the time. According to a consumer survey, a third of TV viewers have given up ever setting a future video recording date and time. Cryptodesign succeeds in maintaining a useless machine interface. The message is clear. Cryptodesign says "a technique useful for one situation is probably good in all situations." The antidote requires that we breath life back into automatic design techniques. Let's call the antidote "soul design".

"Soul design" bridges the gap between technology and user by insuring that we change the technology to meet human needs. Our special field is software and we feel obliged to report a sorry state of affairs among managers and developers. Cultural cryptodesign has the upper hand. For example, we have a CANCEL button on our GUI windows, a de facto standard for English UIs. But which key lets you cancel? The ESCAPE key! The button and the key do the same thing, but cryptodesigners gave them different labels to which we are forced to adapt. By the way, don't press the SHIFT key when you see the ubiquitous message "Press any key to continue." It doesn't work. Users must learn exceptions to the instruction. How do we gear up to eradicate cryptodesign? Developers and product managers must enhance their sensitivity to the work demanded of computer users. This is not a new idea. But developers really make progress when we speak about the four different kinds of work users do. We call it the VIMM model, for Visual, Intellectual, Memory, and Motor work. You can customize your design techniques to reduce these types of work, once you know how to look for them. Here are some examples.

Reduce memory work We found a floppy disk installation instruction that required too much memory work. The instruction on the floppy label reads “Insert the diskette in drive A: and type SETUP”. Then on the screen, the first instruction reads “Key in the code from the label on the diskette, then press the RETURN key.” Yes, you’ve been set-up. You may lose the skirmish. The problem: if you remove the floppy to check it, you must remember to replace it in the drive before pressing the RETURN key! The solution: support the user’s memory by reminding them to “Replace the diskette in the floppy drive. Then press the RETURN key.”

Reduce Motor Work “Motor work” refers to physical movement, like typing. We reviewed a new GUI customer service system for a major credit card service. All their credit card accounts started with the same four digits. So they defaulted the cursor start to the right of the fourth number. A good move. But we learned every call still had four unneeded keystrokes because we pulled in some soul design for reinforcements. Upon dropping our assumptions and interviewing users, we learned that the next two numbers were the same for 99% of callers. So we moved the cursor two to the right and scored a two-stroke reduction. We then avoided tab stops at two objects that were virtually never used, saving another two keystrokes. Those four keystrokes saved over \$10,000 present worth in user time. (Yes, they have a lot of service reps who get a lot of calls.)

Reduce Intellectual Work We often find applications that require more thought and decision making than necessary. (Battles are lost when the troops don’t move.) For example, a large mail order catalog house upgraded its telephone customer service application to a GUI environment. We interviewed and observed the service reps. We learned that the navigation in the current system

failed to handle requests for product returns without considerable thought. First, upon taking the order and getting a customer request to make a return, the rep had to write down the customer ID. Second, the rep “jumped sessions” to the Invoice>Returns module. Third, the rep re-entered the customer ID. Fourth, the rep checked the purchase date for the item and mentally calculated the warranty expiration date. Fifth, the rep exited and returned to the Order module. Sixth, the rep reentered the customer ID. And seventh, the rep entered an entire new order while typing “No charge” to signify a warranty replacement. We improved the navigation by putting an option for “See customer’s warranties” on the order screen. This brought up the customer’s purchase history directly from the order window, saving thought time and reducing errors. Reorder was only a mouse click away.

Reduce Visual Work In the mail-order example, users viewed a scrolled list of purchases for the customer. In the old system, users had to check the purchase dates to establish the warranty period for each item. In the new system, items within warranty were color coded for easy recognition. But this was only part of the cryptocrunch. We also made sure that the gray-value shading was different, too, since about 8% of males have some form of color blindness. (Only about .5% of females are so afflicted). To make it easier to scan across the columns in the purchase history list, we also put a blank line every 5 lines. This provided a “ruler” to guide the eye left to right, preventing scanning errors. By the way, we left-aligned field labels to reduce clutter.

The VIMM model presented a few rules of thumb that you can learn to reduce user work. More will appear in future articles. Next find our two top defense strategies against GUI cryptodesign.

STRATEGY ONE: THROW OFF THE BURDEN OF WINDOWS Be wary of using the “windows” of your GUI environment. Research (and practice) show that end-users can easily spend too much time re-sizing, re-positioning, and selecting windows. The design terminology, “free form windows,” sounds acceptable. But know that the resulting “window thrashing” can make your new GUI application slower than its mainframe equivalent! This is not speculation. It really happened. Users in the credit card company example above rejected the initial launch of their \$10 million GUI system because it slowed them down with too many windows. They were getting paid on a piece basis for handling calls and they hated to take a salary cut, even in exchange for color, mice, and “freedom”. For great examples of window thrashing check out the UIs for CompuServe and America OnLine!

CRYPTOWINDOWS SNEAK ATTACK Here’s how cryptodesign creates such problems. First, many developers and product managers who design for a windows environment feel it’s politically correct to use plentiful “windows.” Finding multitudinous windows in development tools used for designing windows reinforces this assumption for many folks. Ironically, free form windows actually works well for development tools. You can pile up the windows when designing them; but as a computer expert you don’t mind. You need free form windows to juxtapose the various screens for cut-and-paste. Plus you can make visual comparisons between any two windows. Problems arise because cryptodesign suggests that what’s good for the developer is good for the end-user. (It’s probably not true.) However, we find that only 10% of corporate and shrink wrapped software benefit from such free-form windows. This is because free-form windows translates as “lack of structure” in the minds of your end-users. Why

burden users with the designer’s failure to find structure in their UI architecture?

CHECK USER’S NEED FOR “FREE FORM WINDOWS” Here’s how soul design solves the problem. Learn a lot about the user’s task flow. Find opportunities to simplify navigation with techniques that automatically close one work area while opening another one. For example, Microsoft has strongly promoted the “tab” and “folder” metaphor in Microsoft Office dialog boxes. OS/2 has adopted the “notebook” metaphor. Both metaphors solve a window thrashing problem within the lower level dialogs of the application.

The high level UI architecture orients the user to navigating the application. It generally should be the very first screen they get. How do we reduce window window thrashing for the high level UI architecture? For certain task flows, we have placed buttons across the top of the screen, much like tabs. The button row stays up throughout the application, reminding users of their task options. The lower portion of the screen “swaps out” one task for another when the user selects a different button. One button is always “on”, even upon opening the application. We call it the “context switch”. We don’t use the context switch for everything (that would be cryptodesign). First, we interview users and find out what they do. We consider using the context switch if the user must frequently access several screens in a nonsequential pattern. More than five or six such navigation options requires a different approach, since there is room for only five or six buttons across the screen. Note that we resist using icons on those context switch buttons. Cryptodesign would suggest adopting the tool ribbon model given in most word processors—as is done by the unwary. However, we usually recommend buttons with labels because casual users have difficulty interpreting

Software Ergonomics—Its Role Throughout System Development		
System Development Phase	Usability Activities (Simplified)	Consequences of Failure
1. <i>Proposal</i> : Get ideas for a new system.	Performance analysis: Interviews,	Business opportunities are lost, fault isolation, brainstorming.
2. <i>Feasibility</i> : Check that system is practical.	Quantify costs/benefits. Check for simple manual solutions.	An expensive system that is impractical, unnecessary, or awkward to use.
3. <i>Definition</i> : Describe system boundaries, high-level functions.	Gather data on existing environment: user characteristics, task-flow, problems. High-level taskflow design.	A system that does not fit the user or environment. An awkward or confusing system structure.
4. <i>Preliminary Design</i> : Select high-end UI architecture.	Design the screenflow architecture to match the task flow design.	The user jumps around the system to get work done.
5. <i>Detailed Design</i> : Design screen layouts, wording, operation, color.	Standardize screen designs. Standardize error messages. Use protocol simulation testing.	Screens that are hard to understand and use.
6. <i>Implementation</i> : Determine best overall user support.	Prepare user support products: online help, user manuals, job aids, training.	Impractical or unusable documents, training, or job aids.
7. <i>Conversion</i> : Put system in place.	Select and execute best conversion strategy: flash cut, dual run.	An awkward and expensive system installation with lingering bad feelings.
8. <i>Performance Review</i> : Verify that system meets objectives.	Establish testing protocol. Gather data: logs, interviews, probes, questionnaires, analysis and problem resolution.	System with ergonomic "bugs" Developers never learn from their mistakes.

icons! We also avoid toolbar ribbons because they present small mouse pointer targets! But note that for word processing and drawing tools these features are fine because the small icons leave space on the screen for the document.

STRATEGY TWO: BE SOCIABLE Previously we mentioned interviewing users to learn their task flow. For many developers this is old hat, for others it's new. For both however, the danger of cryp-

todesign remains. That is, even though interviews are dutifully obtained, database structures creep into the UI architecture instead of task flow structures. Database structures appeared in the catalog order application mentioned earlier and made life difficult. The user could navigate to the Order module or the Invoice>Returns module, but could not handle a return within an order. Soul design has us sit with users and see what patterns emerge

Interview Technique: How to Conduct a Good Interview	
<p>1. <i>Do Pre-interview Arrangements:</i> Select a spectrum of participants. Get cooperation from the start. Show management support. Establish expectations up front. Be considerate and professional.</p> <p>Method:</p> <ul style="list-style-type: none"> • Send letter from you & management to user and user management. 	<p>2. <i>Create the Setting:</i> Establish a relationship. Get agreements from user.</p> <p>Method:</p> <ul style="list-style-type: none"> • Introduce yourself. • Explain your needs. Show how this helps the user. • Ask user for their help. • Mention interview time requirements. • Explain that user is anonymous. • Disclaim ability to make promises about future software features. • Clarify that this is NOT a "test." • Answer any immediate questions.
<p>3. <i>Create Ice Breaker:</i> Get the user thinking about the area. Get the user talking.</p> <p>Method:</p> <ul style="list-style-type: none"> • Ask 1 to 3 questions (simple, informative and orienting--e.g., "how long have you worked here"). 	<p>4. <i>Ask a Wide-open Question:</i> Get any unexpected responses up front. Get unbiased responses first.</p> <p>Method:</p> <ul style="list-style-type: none"> • Ask a non-threatening question, e.g., "how do you feel about the work you do here?." • Do not permit a one-word answer. • Use the word "feel" to remove the onus of being "correct."
<p>5. <i>Do the Task Analysis:</i> Learn what the user does. Get bird's-eye view, then get details for each section.</p> <p>Method:</p> <ul style="list-style-type: none"> • Work together on paper. • Start with an input, then process, then output. • Gently reject irrelevant points. • Probe for frequency, importance and problems at each step. 	<p>6. <i>Ask Specific Questions:</i> Get specific answers to questions not spontaneously covered.</p> <p>Method:</p> <ul style="list-style-type: none"> • Ask direct questions on issues that came up in other interviews, etc. • Avoid giving the user the answer you want to hear. Let them talk.
<p>7. <i>Ask Final Open Question:</i> Capture the user's final ideas. Do not rush to a close.</p> <p>Method:</p> <ul style="list-style-type: none"> • Ask open questions (e.g., "Anything else we haven't covered?"). • Ask for solutions (e.g., "If you had everything your way, how would you handle X"). 	<p>8. <i>Debrief the Interviewee:</i> Leave the user feeling comfortable. Keep the door open for call-backs.</p> <p>Method:</p> <ul style="list-style-type: none"> • Ask an open question (e.g., "Any questions about this interview?"). • Thank the user. • Ask permission to call back with questions.

from their work. We estimate that 80% of the risk in any project hinges on getting the task flow clearly represented in the UI architecture. You can manage this risk holding back on “screen design” until you’ve finished the first four steps in the accompanying chart *Software Ergonomics: It’s Role Throughout System Development*.

We’re fortunate in getting recent evidence that sociable interaction with users pays off. In the March 1995 issue of Communications of the ACM, Mark Keil and Erran Carmel report that 14 successful projects they investigated had an average of 5.6 different types of “links” between developers and customers. On the other hand, 14 unsuccessful projects had an average of 3.2 such links, with 10 out of the 14 using either zero or only one direct link between developers and customers. The lesson is clear. Soul-designers listen to users. Cryptodesigners listen to themselves. How do we develop direct links with end-users? The interview is the basic technique. Combat requires sharing ideas. We’ve included instructions on conducting a good interview (see next page). Bring it with you to make sure you do it

right. Other sociable techniques include: Joint Application Design (JAD), showing prototypes to customers for comment, testing with prototypes, getting input from users using paper surveys and e-mail, observation, and focus groups. Out of the 15 types of user links mentioned by Keil and Carmel, three required interviews with an intermediary between developers and users. Intermediaries included marketing, sales, and help desk personnel, and user representatives. In each case, be sociable. Ask questions. Find problems—it stops cryptodesign dead.

In future articles we’ll show how to translate the task flow you find into good UI architecture. We already mentioned the context switch as one possible architecture. We’ll give you defense strategies against other cryptodesign temptations such as pull-down menus, icons, and mouse usage! We’ll also show you how to create a useful screen design standard. And we’ll describe the cryptodesign battle in which a large national organization spent \$500,000 on screen standards that were best used as a door stop. Until then, go for soul. Talk to users—and beware of windows!