



HUMAN-CENTERED COMPUTING

Editor: **Robert R. Hoffman, Jeffrey M. Bradshaw, and Kenneth M. Ford,**
Institute for Human and Machine Cognition, rhoffman@ihmc.us

Improving Users' Mental Models of Intelligent Software Tools

Shane T. Mueller and Gary Klein *Applied Research Associates*

Intelligent software algorithms are increasingly becoming a tool in consumers' daily lives. Commercial navigation systems and Web search algorithms are just two examples of how complex optimization algorithms have changed how we live. In many cases, users understand the basic mechanics of the intelligent software systems they rely on, but often (and perhaps more often today than in the past) novices have no direct knowledge of their intelligent devices' algorithms, data requirements, limitations, and representations. Problems can go beyond those caused by a poor user interface design and a user's ability to understand a tool's simple components, which could be alleviated with proper instruction. These problems might be fundamentally cognitive and likely stem from people having an inadequate unsuitable mental model of how a tool works.¹

Usability problems can persist even for software designed with human-centered design principles, especially when the software replaces or augments intelligent human capabilities. In these cases, users might have incorrect expectations about what the automated system is doing, and training and guides are necessary to help them understand the software's workings. Usability problems can often be overcome by trial-and-error experience or when communities pass down information to newer users, but we believe that the genuine cognitive challenges to forming functional and accurate mental models can be formalized, documented, and "trained in." This article describes the Experiential User Guide (EUG), a concept designed to address these challenges.

Background and Rationale

A person might misunderstand several aspects of a software tool:²

- its data requirements;
- its representation and modeling mechanisms;
- its underlying computations and algorithms;
- the appropriate range of situations for which it applies; and
- the output, display, and visualization it provides.

These categories cover a fairly broad range of software systems, but they help scope the type of tools that an EUG might be appropriate for. Software tools that don't use intelligent software techniques in one or more of these areas can still be difficult to use. Our intention is to identify guidance and training materials that are especially useful for these functions, particularly when the functions are not transparent to the user. As an example, a word processor's mail-merge function might be difficult to use, but its function involves no genuine intelligence, so it is not the type of function we feel is best targeted by an EUG. In this case, the mail-merge functionality should probably just be redesigned.

To gather ideas for the EUG, we looked at a range of experiential training and support methods that have been used in the past, both for software and nonsoftware applications. One useful place we looked was the computer games domain, especially gamer-developed training. Games often have complex visualization and intelligent agents, which make them reasonable targets for the EUG approach. Most computer games with substantial user communities have forums and websites (hosted either by the game developer or third parties) that offer strategy guides, walkthroughs, and sometimes in-game scenarios to help novices learn the system.

From examining such support tools, we learned several things. First, experiential training does

not require interactive multimedia and can be fairly simple; many gaming strategy guides do not even contain screenshots of the game and rely completely on text. Second, the training need not be tightly coupled to the tool. Although training embedded within a game or tool can be effective, many third-party and user-generated strategy guides are stand-alone, most likely because substantially more effort can be required to create training embedded within a tool than to create an independent training material. Third, walkthroughs are commonly used in gaming contexts, but they could be more widely adapted and used in nongaming as well. Just as a walkthrough gives a gamer instructions on the steps needed to complete a level or game, it can similarly be used to give a user a simple set of steps needed to complete an analysis or build a model.

We also looked at many guides intended for crafters and do-it-yourself enthusiasts and hobbyists. Although do-it-yourself guides have existed for decades, the newest generation of these includes many how-to websites (such as makezine.com, wikihow.com, instructables.com, and ehow.com) that are fundamentally different from previous guides. These usually focus on a single, project-based topic; examples include

- “Reuse your walkman’s shell to hold an iPod” (http://blog.makezine.com/archive/2008/02/reuse_your_walkmans_shell.html),
- instructions for an apartment-friendly “Stolmen bike rack” made with IKEA parts (www.ikeahackers.net/2008/02/stolmen-bike-rack.html), and
- “Make a Valentine’s Day sock monkey!” (www.instructables.com/id/Make-a-Valentines-Day-sock-monkey).

These typically have step-by-step instructions and include photographs or even video of someone doing the project. They rarely include more than rudimentary plans, such as blueprints or custom graphics that pervade popular-press do-it-yourself guides. These how-to websites can also succeed by being low-tech because highly interactive content and video cannot be printed and used as a reference.

These types of websites range in their usefulness as a model for supporting intelligent software tools in experiential ways. They embody important aspects of experiential training because they encourage people to learn how to create a project or solve a problem by doing it. Nevertheless, most of these guides focus on a single pathway to making a project, without helping the user understand alternatives, workarounds, or potential errors that we think are important for an EUG. Furthermore, they are not intended to teach the reader how to use the tools required to complete the project—this learning would only come as a side effect.

How-to guides are similar to recipe books, which we also considered. Such books typically give abbreviated task instructions, sometimes with variations or workarounds for when ingredients are unavailable, but they assume a certain level of familiarity with cooking in general. For example, one family recipe for peanut butter cookies reads:

Peanut Butter Cookies

*1 cup granulated sugar
1 cup brown sugar
2 eggs
2 cups chunky peanut butter
Drop by spoonfuls on cookie sheet and
bake about 10 minutes at 325°.*

The recipe is beautifully terse, but it tells nothing about the bounds of the recipe. Can it be made with two cups brown sugar, or can you use white sugar alone if you add molasses? What size eggs should be used? What if you use unsalted peanut butter? What if you want the cookies chewier? It is left to the baker to experiment or improvise to answer such questions.

Recipes (and indeed, most engineering design documents) tend not to provide the rationale behind and intent of the instructions or ingredients. One ingredient might be used for taste, whereas another might provide leavening. Also, you might bake a recipe instead of frying to improve its texture. The reasoning behind these choices is rarely discussed, so workarounds can be difficult to determine. The same thing tends to hold for how-to books and videogame walkthroughs—they provide a minimal path to solving a problem, which allows for experiential training in building, cooking, or gaming, but only indirectly.

From these sources, we identified several concepts for EUG modules as well as methods to avoid. A recipe card, a how-to guide, or a gamer’s walkthrough might be ineffective if it does not also help illustrate boundaries and variations that could work as well. In other words, they can encourage a brittle understanding of the tool.

One final type of training guide we identified is probably most similar in spirit to our notion of an EUG: writer’s guides. These include mechanistic or rule-based guides, such as ones that explain how to format a screenplay or how to develop character arcs, scenes, and plot points (such as Dana Singer’s *Stage Writer’s Handbook*³). Others are knowledge-based guides, such as Serita D. Stevens

Table 1. Methods for developing an Experiential User Guide.

Empirical method	Opportunities and strengths	Cautions and weaknesses
Examining web-based forums, FAQ documents, user-edited wikis, and bug report databases.	These methods sometimes include questions and answers regarding the tool's use and types of problems and misconceptions users have had.	User-generated problems can arise from many sources, so information must be triaged to focus on problems that stem from cognitive mismatches between the user's mental model and the tool.
Incident-based interviews with system developers and trainers.	System developers can have a rich and (often) accurate understanding of how the tool is supposed to be used. They also sometimes have experience teaching others, and training strategies can be gleaned from their experience.	Developers can find it difficult to understand why novices have trouble grasping the system from the user interface alone. Developers might create interfaces and functions that they themselves never employ to do the cognitive work. Developers can be responsible for deep aspects of the system and might never have seen the interface, and others employ backdoor methods to access the program.
Documentation analysis.	System design documents and academic papers sometimes provide a picture of the mechanisms underlying a tool.	Documentation typically will not provide much insight to a novice's incorrectly specified mental models.
Examining expert- and novice-user-developed models.	Some tools produce records of their use, including models, data logs, reports, or simulation output. These can help identify places where novices fail and experts succeed.	The end state of an analysis or model might not represent the many challenges overcome during model building and testing.
Observing a novice's first session.	Observations can reveal aspects of the tool that experienced users incorporate into their mental model early on and then forget about, especially aspects related to properly using the tool and its boundary conditions.	Observations might focus on superficial problems or misconceptions that only happen during the first contact with the system.
Examining notes taken as an EUG developer learns the tool.	Notes about issues, misconceptions, and problems as they arose facilitates the creation of EUG lessons, at every level from creating screenshots to crafting problems that highlight the concepts of interest.	This approach might not be possible for all tools. For example, tools that are embedded within a physical system might permit only limited access to the EUG developer. A member of the EUG development team must have the ability to precisely understand the tool's underlying mechanisms and algorithms. Furthermore, they must be partnered with others who understood fields such as human factors and instructional design.
Observing training of small groups of new users.	Important learning objectives can be identified and tested on a group of trainees. This reveals conceptual challenges that novices face and allows testing of EUG concepts and getting immediate feedback.	Training will likely have at least some goals that differ from those of the EUG, so lessons learned through training might not transfer.

and Anne Klarner's *Deadly Doses: A Writer's Guide to Poisons*,⁴ which explains the practical theory behind poisons and provides descriptive accounts to help writers create compelling stories that use poisons. These guides differ from typical software guides for word processors in the same way the EUG should differ from the basic function-based help files that is typically included in software. That is, they show how to use the software to accomplish a complex goal, using narrative experiential scenarios.

EUG Data Elicitation Strategies

An important first step in developing an EUG for a system is to identify the specific gaps in understanding that

can be translated into learning objectives. An EUG developer should begin by examining the main functions (data, representation, algorithm, and output). Other constraints to consider are whether the EUG team has access to the tool (some embedded or classified systems might not be available for exploration), whether novice and expert users are available for interviewing (neither might be for tools in development), whether system developers and training personnel are available, whether existing models or analysis output can be obtained, and whether a user community exists. These constraints will typically require using a number of complementary approaches to identify learning objectives. Table 1 describes some possibilities.

For two different EUG tools, we conducted incident-based interviews with system architects, software developers, mathematicians, and secondary developers responsible for testing and validating the tools.⁵ Our goal was to gain an understanding of the developer's mental model of the tool. These often took the form of high-level analogies—for example, a system dynamics model of population might be thought of as a hydraulic system of stocks and flows and a Bayesian network might be viewed as an abstract sketch of a complex interactive system.

The concern is that expert users, like system developers, can have their thinking captured by the tool. Experienced users tend to think

Table 2. Types of Experiential User Guide learning modules.

Module	Description	Rationale
A wizard or walkthrough.	Gives generic advice on how to use the tool properly for a new problem.	After watching step-by-step instruction, many users are lost when it comes to repeating steps on a new problem.
Forced-choice scenarios.	Presents a scenario and a comparison of two alternative solutions.	Shows positive and negative usage quickly with low overhead.
Induce user error, or troubleshoot error in another model.	Presents a scenario with a specific problem, error, or misinterpretation, and allows learner to discover the problem.	Errors highlight boundary conditions and problem areas.
Give assignment, and then show an expert solution.	A scenario is presented, and the learner builds a model and compares it to expert solutions.	Examples of proper solutions to already-worked problems can help users learn a tool.
Work a problem with the tool to answer a question about how the tool works.	Nontransparent functions of the tool can be learned by asking learners a question about operation and having them determine the answer via tool use.	Expertise develops through experiences when users discover how a tool accomplishes some task or thinks about a problem.

about situations within the tool's vocabulary (and buttonology) and can even find it difficult to identify situations for which the tool would be inappropriate. We have interviewed users who were explicitly aware of how their reasoning had been captured by the tool in this way. It is remarkable how a novice's experience can differ from the experts; a novice might have a hard time finding a problem that a tool can be used for, while an expert has a hard time thinking about a problem to which the tool could not be applied.

We found it useful to collect artifacts from expert and novice usage (which might include models, simulation results, reports, and the like). For one modeling tool we examined, we were able to obtain approximately 25 models that had been developed by both novices and experts. Examining these models showed us many of the typical practices of experts, as well as novice errors, and sometimes even expert errors. For example, one tool we examined required the user to create a causal model by drawing box-and-arrow diagrams. One novice user began treating it like other similar tools he had used previously, essentially creating a concept map. This was an improper use of the tool that ultimately created an invalid model because of the user's miscalibrated mental model of the tool—in

this case, the meaning of the linking arrows. Thus, users might need to explicitly unlearn habits and inappropriate mental models from other tools before they can use a new tool successfully.

The effectiveness of identifying a tool's boundary conditions has its limits, which we learned from observing a group of trainees. We showed trainees examples of improper usage of a particular tool, which was helpful for identifying boundary conditions, but it also became demotivating because it highlighted many of the ways the tool could be misapplied. Thus, negative examples must be balanced with examples of proper use to improve the chances for learning.

EUG Modules

The goal of an EUG is to provide reference and training guidance for navigating the cognitive challenges of intelligent software tools. Because intelligent tools perform complex operations that many people are incapable of comprehending or unwilling to understand analytically, we believe the best method to improve their knowledge of the tool is through experiential training. In our research, we identified several types of modules that can place a novice in the same types of situations that experts face to help them refine their mental

model of the tool. In general, these modules include examples of how the system can be used successfully and illustrations of how it can be misused to help users understand the system's boundary conditions. Table 2 describes some of these module types.

Example 1: Java Causal Analysis Toolkit

As we first developed the EUG concept, our focus was a Bayesian analysis tool for intelligence analysts developed by the Air Force Research Laboratory called the Java Causal Analysis Toolkit.⁶ JCAT is a modeling tool that enables analysts to create time-based scenarios of causally related events. It is fairly abstract in that it is not intended for any particular domain. Events within the model (represented as network nodes) are labeled and parameterized by the user to map onto some hypothesized real-world process (in the user's domain of expertise). An example use case might be in the medical diagnosis domain, where a causal model of disease risk might be constructed (with causes related to a base rate associated with health history, genetic profile, environmental factors, and behavior), and an individual patient's test results might be used as evidence in the model to calculate their risk of disease. We developed an initial EUG for JCAT with approximately 10 modules that covered many

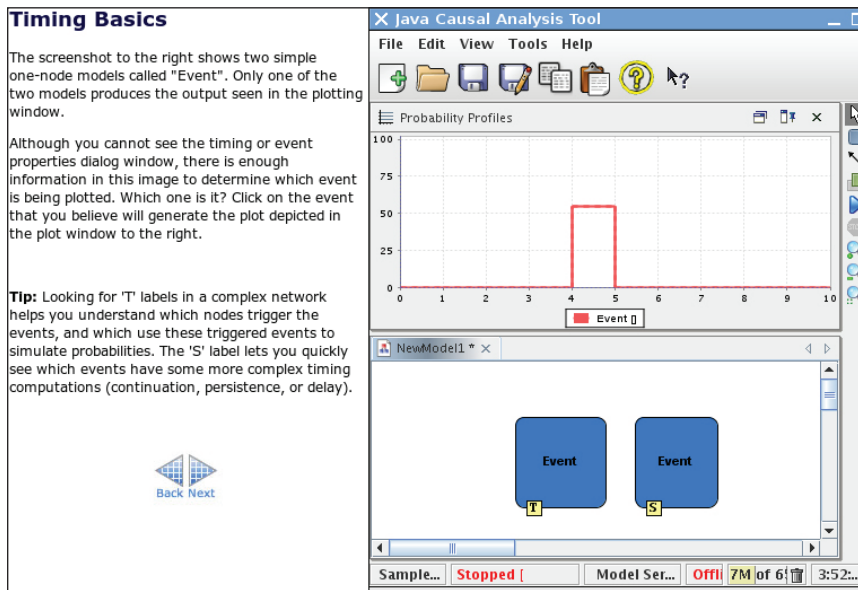


Figure 1. Screenshot of an example Experiential User Guide lesson. The forced-choice scenario demonstrates an underlying modeling paradigm that previous novices had problems comprehending.

of the mistakes and misconceptions we identified.

We created one EUG module for JCAT because of a common mistake we saw: a user would create a branch of a causal model leading into an outcome event, but it would have no effect on the outcome. This arose because users failed to understand that, generally, an event could only occur in two ways: the event could be caused by another event (represented as a link between nodes), or it could cause itself. Similar to a chain of dominoes, if the first “trigger” node does not occur, the entire branch is essentially dead and can have no impact on the model output. Experts understood this, and we observed how they examined the display to determine whether a branch of nodes was active or inactive (by identifying a yellow tag on the trigger nodes). We incorporated this lesson in our EUG, which helped support the user’s understanding of the difference between trigger and nontrigger nodes. Ultimately, this produced a better calibrated mental model of JCAT.

The setup for this problem-based exercise is simple. Figure 1 shows the

first page of our EUG lesson, a JCAT network with two nodes that produces a single output. After reading the initial problem, the user is asked to select one of two models that produced the output. The only difference between the two nodes is the letter on the yellow annotation, with one (S) indicating a scheduled trigger node and the other (T) a timing property. An incorrect choice demonstrates that the learner does not yet understand the diagram’s meaning and is unlikely to know the model-building concept of trigger nodes. After a choice is made, rationale-based feedback is provided. If users gave an incorrect choice, the lesson gives them a chance to correct it. Overall, the scenario helps novices learn about both the interface and the underlying modeling paradigm that many previous novices had missed.

Example 2: A Routing System

Sometimes a novice will see the output of an automated system and interpret nonintuitive results as a signal that the system is broken or untrustworthy. An experienced user, on the other hand, might look at the same

output and see the behavior as a signature of some underlying algorithm, and thus see it as evidence the system is working properly. We have witnessed this in several contexts, and it typically represents a difference between novice and expert mental models of a system. Thus, we leveraged this in another EUG lesson.

This situation arose for a software tool that planned a route to an objective across terrain. The system finds an initial route that the user can then adjust by placing avoidance zones on the map, telling the software to plan a route around a region. The system’s response to some of avoidance zones could produce results that might puzzle novices but reassure experts. That is, the same behavior is taken by one group as evidence the system is broken, and the other as evidence the system is working.

Figure 2 shows a contrast set EUG lesson to support understanding of this function. In this case, option B might be more intuitive to the new user because it produces a more direct route. However, the routing system might be more likely to produce the option A route because it is attempting to remain closer to the original route, perhaps because of other constraints. This EUG lesson attempts to help a novice gain an expert-level knowledge of the tool.

What Makes for a Good EUG?

An important aspect of the EUG is that it helps a user understand the boundaries of proper and improper tool use. Just as the EUG delineates the bounds of a tool, we can apply this same lesson to delineate the bounds of an EUG.

Intelligent Software Functions

An EUG is for nontrivial intelligent software functions. Experiential

training has proven useful in many contexts, so it is certainly true that experiential training could be effective for many ordinary software functions. We focus on intelligent software functions because users are especially apt to need an accurate mental model of those systems, while at the same time nonexperiential methods for training users in the system are likely to be highly technical and beyond what most users will tolerate. For straight-forward software functions, scenario-based learning might be less efficient for users, because they might simply want to look up a function reference in a detailed help system. In addition, limited resources mean that investments in training need to be prioritized.

Thus, it is reasonable to focus experiential training on the aspects of the tool that are either most likely to benefit from the training, or most likely to cause problems if there is no experiential training. So, although an EUG could be created for many non-intelligent software functions, we advocate its use primarily for intelligent functions that present genuine cognitive challenges to users.

User Experience

An EUG is not an introductory course book. When designing an EUG, developers must gauge a typical EUG user's experience. This will depend on the tool, the user population, and the existence of other training manuals and courses. When identifying learning objectives, a decision must be made about whether a training lesson really belongs in the guide or if it should be handled via other means.

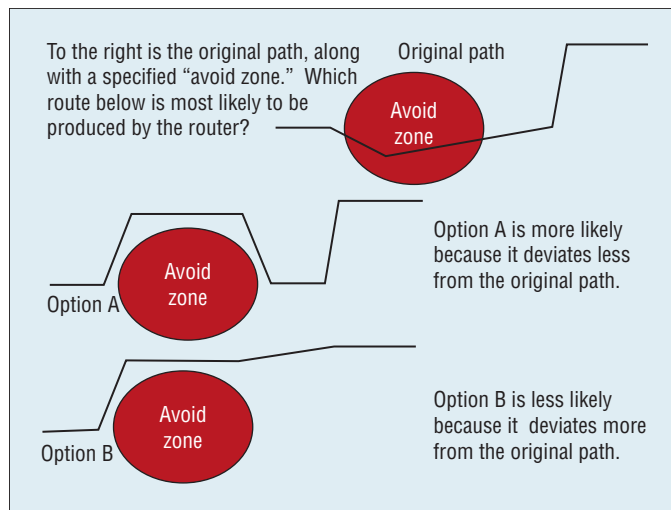


Figure 2. An Experiential User Guide lesson to support the interpretation of the tools' status. In this case, if a user identifies an explicit zone to avoid, a routing system might produce option A, which could seem incorrect to the novice (because it represents a greater deviation from the shortest path) but correct to the expert (because it deviates from the original path the least).

For the systems we examined, there was little formalized introductory-level training material, which posed a dilemma; the EUG was intended to focus on developing a functional and accurate mental model of underlying intelligent algorithms, but if users have no way of understanding a tool's basic functions, they might never get to the point where a calibrated mental model matters. We suspect that this issue will need to be balanced any time an EUG is created.

Usability

An EUG should not replace usability testing. Usability concerns are almost always a huge challenge for special-purpose software with small user bases, for which we also anticipate the EUG being the most valuable. The process of developing an EUG can unearth many usability issues, but we recognize that there is a distinction between the kinds of usability issues revealed by standard usability testing and the types of problems the EUG should focus on. Usability testing is often predicated on the notion that the interface can

be made easier to use. The EUG should support the types of functionalities that cannot be changed, cannot be made easier, and will necessarily require experience and training to get right. By analogy, user testing might help someone design the ergonomics of new musical instrument, but it will never make the instrument so easy to play that even a novice can perform masterfully.

An EUG is created specifically to accelerate proficiency in intelligent software tool use. It exposes a learner to many of the experiences that an expert will have had over time, allowing the learner to experience the tool's strengths and weaknesses. It is intended for software tools that perform nontrivial or intelligent functions and can ultimately help both novice and experienced users gain a new and better understanding of their tools. ■

Acknowledgments


This article partially describes work sponsored by the Air Force Research Laboratory Human Performance Wing that was originally performed under subcontract number TACS-SC-04-144, TO 0013, from Northrop Grumman IT (prime contract FA8650-04-D-6546). We thank Craig Stansifer, Janet Miller, and Cheryl Batchelor with AFRL/RHCF for encouragement and guidance; Chris Burns for designing and implementing EUG modules; and Oliver Tan Kok Soon and the Singapore Defence Science Technology Agency for sponsoring a workshop on the EUG that helped focus the method.

References

1. P. Koopman and R.R. Hoffman, "Work-Arounds, Make-Work, and Kludges," *IEEE Intelligent Systems*, vol. 18, no. 6, 2003, pp. 70–75.
2. S.T. Mueller, G. Klein, and C. Burns, "Experiencing the Tool without Experiencing the Pain: Concepts for an Experiential User Guide," *Proc. 9th Int'l Conf. Naturalistic Decision Making*, British Computer Soc., 2009, pp. 105–112; www.bcs.org/upload/pdf/ewic_ndm09_s1paper11.pdf.
3. D. Singer, *Stage Writer's Handbook: A Complete Business Guide for Playwrights, Composers, Lyricists and Librettists*, Theatre Comm. Group, 1997.
4. S. Stevens and A. Klarner, *Deadly Doses: A Writer's Guide to Poisons*, Writer's Digest Books, 1990.
5. B. Crandall, G. Klein, and R.R. Hoffman, *Working Minds: A Practitioner's Guide to Cognitive Task Analysis*, MIT Press, 2006.
6. J. Lemmer and B. McCrabb, "A Genuinely Bayesian Tool for Managing Uncertainty in Plans," 2005; <http://uncertainreasoning.com>.

Shane T. Mueller is a researcher at the Klein Associates Division of Applied Research Associates. Contact him at smueller@ara.com.

Gary Klein is a researcher at the Klein Associates Division of Applied Research Associates and at MacroCognition. Contact him at gary@macrocognition.com.

 *Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.*