Experiencing the Tool without Experiencing the Pain: Concepts for an Experiential User Guide

Shane T. Mueller

Gary Klein

Christopher Burns

Applied Research Associates, Inc.

smueller@decisionmaking.com

ABSTRACT

Motivation – Intelligent software tools face substantial obstacles to adoption, because of their complexity and lack of transparency. **Research Approach** – We describe the *Experiential User Guide* for intelligent software systems, which aims to compress critical challenges users discover along the pathway to expertise. **Design**– The EUG uses vignettes to illustrate learning challenges, employing examples of how the tool can be used successfully, and examples of its limitations, boundary conditions, and the common errors made by users of the tool. **Research limitations/Implications.** The EUG focuses on advanced concepts of operation, as these are areas that can especially benefit from experiential training. **Originality/Value** – The EUG shows how software is used *in situ*, rather than the more typical tool-centered approach describing the different functions of the software system. **Take away message** – The EUG is a new approach to help users of intelligent software tools become experts faster and more systematically.

Keywords

software user guide, experiential learning.

INTRODUCTION

Our goal in this paper is to describe components of a new concept for supporting users of intelligent software tools and other complex technology systems: the **Experiential User Guide (EUG)**. We will first describe some of the rationale for the EUG, including the types of systems it can be most useful for. Then, we will report on some of the generic components that make up an EUG. Finally, we will present highlights of an EUG we are developing for a Bayesian reasoning and modeling system called JCAT (Java Causal Analysis Toolkit).

BACKGROUND AND RATIONALE

There are many ways software developers document their tools to provide help to users. In the early days of personal computers, most software came with one or more heavy bound manuals which documented all of the features, shortcut keys, screens, and uses for the tool. As desktop computers have increased in power, these have been supplanted by help systems embedded in the software itself. These offer context-sensitive help, search, indexing, and tighter integration with the software tool, and often cost less to produce. Many embedded help systems document how particular buttons or functions of a software tool work, but leave out critical training about how to use the tool to solve typical and complex problems that users face. Sometimes, more advanced books and training is available for popular software titles, often developed by third party vendors. These more advanced training and user guides use a broad range of metaphors and strategies, many of which we examined to help develop the EUG concept.

To understand the properties that an EUG should have, we evaluated approximately 30 types of user guides, documentation systems, and tutorial approaches used to help novice and advanced users of a very broad set of systems. Our goal was to identify aspects of user guides that are particularly successful in their domain, in order to inform the design of our EUG. Table 1 summarizes some of the highlights and lessons learned from this evaluation.

One lesson we learned is that a user guide need not be tightly integrated into the software tool it supports in order to be successful. Many of the systems we looked at were developed by third-party sources, and often generated by users instead of the software developers or by technical writers. For example, on-line gaming communities frequently have large numbers of user-generated documents containing tips, strategy guides, walkthroughs, cheats, etc. These gamers typically cannot tightly integrate their help with the gaming software, and even in cases where this is possible (e.g., for games that allow users to generate their own content, levels, or scenarios), the skill and effort required to do so can be substantial, which also presents an obstacle that makes lower-tech alternatives more attractive.

Another source of user-generated help comes from “howto” websites, which offer tutorials for do-it-yourself projects (examples include: *How to reuse a Walkman shell to hold an iPod*; *How to build an apartment-friendly bike rack*; *How*

to make a Valentine's day sock monkey). These typically have step-by-step instructions and include photographs or video of the author doing the project. These howto websites also succeed by being low-tech, because highly-interactive content and video cannot be printed onto paper and used as a reference. These tutorials are also project-based, and so they offer an engaging experiential lesson in using the tools needed to accomplish the project.

System	Examples	Unique Features	Comments
"Howto", "Hack" and "Maker" websites	Instructables.com, makezine.com, hackzine.com	Project-based, step-by-step how-tos and tutorials.	Highly experiential, focused on one-time projects.
O'Reilly Hacks book series	Google Hacks	Comprehensive set of advanced tips and tricks about specific tool/software/website	Provide a comprehensive set of tips and tricks for a single tool or platform.
Gaming: In-game tutorials	Second Life's "Orientation Island".	Experiential set of tasks to understand system functionalities, tightly-integrated with automated mentoring.	In-game tutorials blur line between learning and playing, making them an attractive experiential guide.
Gaming: user-generated strategy guides	heavengames.com	Typically exists outside the gaming system; include screen-shots, screencasts, or text-based descriptions. Often task-based, challenge-based, or level-based.	Does not require tight integration with software; deals with learning strategies to succeed in a complex environment.
Gaming: Walkthroughs	heavengames.com	Detailed shortest-path set of steps for accomplishing goal.	Can provide just-in-time support for tough problems.
Miniature Golf	Tognazzini's <i>Gentle learning curves of miniature golf</i>	Step-by-step ladder of challenges requiring skills of increasing complexity.	Natural sequences of lessons in EUG should be supported, but not required
Recipe Books	<i>The Joy of Cooking; Mastering the Art of French Cooking</i>	Basic guides for using a tool (the kitchen) to perform a task (create food).	Cookbooks range from targeted to general; are sometimes experiential.
Writer's Guides	<i>Stage writer's handbook; Deadly Doses, A writer's guide to poisons</i>	Tells how to use a tool (word processor) to solve the problem (take plot ideas and create believable narrative)..	By analogy, a writer's guide is to a word processor manual what the EUG is to a typical user's guide.
Introductory-level instruction books	<i>Chess for Dummies, Complete Idiot's Guide to string theory, Excel for starters: The missing manual</i>	Covers basics in a friendly way; does not alienate learner.	Typically targets highly-used consumer software with focus on most basic instruction.
Computer Help systems	Clippy, MSDN help.	Embedded, context-sensitive, and can be comprehensive and include examples.	Implementations usually fail to live up to their potential, focusing on features rather than accomplishing tasks.
Automated Tutoring Systems	Circsim tutor (Evens & Michael, 2006)	Detailed analysis of whether learner understands complex set of causal relations.	Simulates some of the role of a 1-on-1 mentor.

Table 1. Types of user support systems we evaluated to identify the properties of an EUG.

Another lesson we learned is about the scope of different help systems. Some provide a comprehensive guide for advanced use of a tool or system, while others are targeted to a particular use, or a particular user set. For example, some cookbooks (e.g., *The Joy of Cooking*) aim to provide a comprehensive set of recipes, so that there are several options for almost any type of food the cook has available. Others target specific types of recipes (Julia Child's *Mastering the Art of French Cooking*) or specific types of users (*The Starving Student's Cookbook; The Everything Kids' Cookbook*). Similarly, user guides often target novices (e.g., *Chess for dummies* and similar books in the *For*

Dummies and the *Complete Idiot's Guide* series), providing detailed information to allow the user to gain enough facility to discover advanced uses on their own. Our belief is that an EUG should not try to be a comprehensive guide about the tool; rather it should focus on the places where there are genuine cognitive challenges for tool use.

Other help systems provide a learner-based organization, focusing on the sequence of tasks needed for achieving proper understanding. Tognazzini's (1998) example of how a miniature golf course provides its own user guide is an excellent metaphor: early holes exercise simple skills, which then get combined in increasingly complex ways. Similarly, automated tutoring systems such as the CircSim tutor (Evens & Michael, 2006) which trains medical students about the causal relationships between different aspects of the circulatory system, provide more adaptive methods to deliver the right lessons to the learner at the right time, while simulating some of the role that one-on-one tutors can provide. From these we learned the importance of identifying a sequence that the user should ideally use the EUG.

What types of systems is an EUG good for?

Although experiential training can be an effective way to support the learning of any complex task (e.g., learning to grow a lawn, or to drive a car, or to use a spreadsheet), there are some bounds on the types of tasks and systems we will focus on. Our focus is on intelligent software tools that pose real cognitive challenges to users. These are systems that perform non-trivial reasoning, so that part of developing expertise with the system is understanding this reasoning. Potential users who do not understand this reasoning can become demotivated and can begin to distrust the results of the system or their own ability to use the system. This can lead potential users to reject a system that might actually be useful to them. We have identified four major aspects of systems that may pose non-trivial cognitive challenges to users: representation, data, computation, and output. These are defined and described in greater detail Table 2.

Function	Definition	Examples
Representation and Modeling	How the system represents information, or how the user must generate a model of the world	Networks, vector spaces, trees, flowcharts, agents, rules.
Data Handling and Data Generation	How input for the system is generated.	Probabilities in Bayesian models; text corpora, sensor reliabilities.
Understanding Computation and Algorithms	How input are transformed using computational or mathematical processes.	Singular-value decomposition, Bayes rule, machine learning algorithms, fusion algorithms, optimization routines.
Output, Display, and Visualization	How the results are presented to the user to allow inference.	Visualization methods, tables for comparison; numerical scales, categorical classification (red-yellow-green).

Table 2. Four primary functions of intelligent software tools.

Different tools and systems rely on different subsets of these functions. For example, a modeling tool will focus mainly on **representation**, and the primary cognitive challenge for the user is understanding whether the system's representation is similar to or different from his or her own conception, (and to some extent, whether it is similar to or different from how such a system would work in reality). **Data handling and data generation** differ across systems as well. For example, systems that fuse multiple data streams may require an understanding of the raw data that is hidden from the user. Without confidence about the source of data, users may dismiss the results as mysterious; and without proper skepticism in the limits of the data, users may place unwarranted trust in the results. Understanding the non-trivial **computations and algorithms** a system uses is also critical to allow confident tool use. For example, a GPS navigation system performs a "black box" optimization that is often opaque to users. Whether a user trusts the system's chosen route can depend on their own understanding of the geography, traffic patterns, time of day, and so on. Part of gaining expertise in a system is learning to understand how underlying algorithms work, and if the system often produces unanticipated output, a user might begin to lose trust in the system. Yet part of the need for such systems is because people are unable to perform the processing on their own. This conundrum is a prime motivation for the EUG, as the EUG can help identify the boundary conditions where the result of the tool should be given less credence than others. Finally, **output and display** can be both an inhibitor and facilitator of tool understanding, trust, and adoption. Visual displays often rely on the human visual system's ability to make sense out of patterns, which can be helpful, but can also be misused (as argued by Tufte, 2001).

What does an EUG consist of?

The goal of an EUG is to compress many learned lessons users face along the pathway to expertise. These include experiences that both show how the target system can be used successfully; and experiences that illustrate the boundary

conditions of the system, expert workarounds, and typical errors that users might make. Descriptions of some of these components appear in Table 3.

Name	Description	Rationale
A 'wizard'	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 examples quickly with low overhead
Troubleshoot 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; this provides experiential support of that situation and its resolution.
Give assignment; see expert solution.	A scenario is presented with adequate description, and the learner builds a model, then compares to expert solutions.	For modeling tasks, a tightly-scoped scenario can show range of valid solutions to problem. Multiple valid solutions can be valuable.
Give assignment, show steps expert took to solve problem	A scenario is presented with adequate description, and the learner builds a model, then watches the steps an expert uses to build model (with rationale)	In many cases, the individual steps taken are as important as the final result.
Make error and show learner how to detect and avoid it..	A scenario is designed that compels learner to experience common error representing a boundary condition; resolution provides understanding of where the boundary is.	Novices are often not aware of their errors immediately; this helps them experience the errors and identify ways to avoid them in the future.
Work a problem with tool to answer a question about how the tool works	Non-transparent functions of the tool can be learned by posing a question about operation to the learner, and having him or her determine the answer via tool use.	Expertise develops with experiences where user discovers how a tool accomplishes some task or "thinks about" some problem.

Table 3. Types of learning modules that can comprise an Experiential User Guide.

Example implementation: An experiential user guide for JCAT

The first software system we chose to examine in service of developing an EUG was a Bayesian modeling tool called JCAT (Java Causal Analysis Tool; see Lemmer & McCrabb, 2005). JCAT is a software tool used to build and analyze causal networks, primarily for analysis and planning in military situations. It is developed and distributed by the Air Force Research Laboratory in Rome, NY. It is typically used for building models of causal relationships between events, in the form of node-and-link networks. It simulates the probabilities of these events based on the assumptions the modeler makes, and can perform Bayesian inference regarding the posterior probabilities of events given observed evidence. It offers several unique advances over similar tools, reducing the complexity of the models that need to be created, and allowing time-based relationships to be used in the system in fairly simple ways

Figure 1: Screenshot of JCAT, illustrating a model (network at bottom) and probability profiles of different events (line graph at top).

Basic Objectives	Descriptions	Sources
Representation and Modeling		
Minimal requirements for valid model	Nodes, mechanisms, signals, and triggers are necessary	Novice models
Causation does not imply timing delay	JCAT allows causation within time steps, without expected delays	Analysis of tool components
Role of conditional probability	Confusion in novices of whether it is for outgoing links or incoming links	Novices users
How to model event drivers	Leaks versus external event versus scheduled event.	Novice errors
Leaks are 'unmodelled externalities'	Leaks should be reserved for 'other causes'.	Discussions with developers
Trigger Nodes	Model should be 'driven' be trigger nodes; external factors in the model.	Developers/instructors
Network causal metaphor	Mechanism versus signal: a signal is like an electric current in a wire.	Interview with developer/trainer
Timing	Basic survey of the timing capabilities of a single node	Analysis of tool components
Basic Network Concepts	Nodes are EVENTS; links are causes	Interview with developer/trainer
Causal Loops	Tricks are needed to make a proper causal loop.	Analysis of tool components
Proper definition of an event	A node can be mis-defined to represent event you are not interested in.	Developers & model building
Causal versus Inhibitory impacts	Related but distinct from the inverted signals (above)	Analysis of tool components
Creating growth	At least two methods: invert a decay, or add persistence to a node/leak.	Novice users
Level of Abstraction	How to choose the 'right' level of abstraction	Developers & Novice errors
Does model match user's concept?	Does representation in tool match what the modeler believes it to be?	Discussion with developers/users
Data Generation and Data Handling		
Probability Estimation Errors	Overestimation, hard/easy effect, use of heuristics & biases	academic literature
Testing whether errors matters	Tricks for testing sensitivity of outcome to variations in inputs	Developers/instructors
Need to understand modeling domain	Domain understanding avoids many mis-steps.	Developers; novice observation.
Extent of match between model and real-world process	Models necessarily are not identical to the process they represent; a user should understand the matches and mismatches.	Discussion with modelers
Inability to estimate probability	Inability to estimate probability is a hint that the model is framed poorly	Developers and trainers
Understanding Computations and Algorithms		
Timing overrides everything	Putting a 'scheduled' timing event overrides external influences.	Developers; model-building.
Nodes really are events	It is an error to interpret a node as something other than an event.	Developers and users
Cautions for multiple paths	Care must be taken to produce the correct intended effect.	Discovered while using tool
Role/Use of sampler	Basic description of how it works.	Observation of novices.
Probabilistic events versus causes	Identical outcomes, but making wrong choice can create problems.	Developers; model-building
Meaning of OR	Basic notion of a 'weakest link' model.	Analysis of tool components
creating an AND gate	Combining multiple signals to form an AND rule.	Analysis of tool components
And & Or together	Non-intuitive effects when and and or concepts are combined.	Discovered while using tool
Common causes of events	Related to prob/deter events/causes.	Developers and model building
Use of Evidence	Important feature of tool that creates Bayesian logic.	Analysis of tool components
Gates with On/Off logic	Use of inverted signals, and how they can create exclusive logic.	Analysis of tool components
Conditional Probability Logic	How events are influenced causally by other events.	Observation of novice user
What-if games	Ways to 'pulse' a model to discover ways to modify it.	Interviews with developers.
"Sopranos model" misconception	Misinterpretation of probability profile to be probability distribution.	Developers and users.
Output, Display and Visualization		
Basic Network Concepts	Nodes are EVENTS; links are causes	Developers and trainers
Networks JCAT is not good for	PERT, playoff charts, social networks, org charts, etc.	Developers & users.
Probability Profile	Meaning of probability profile, distinction between "when something happens" and "how likely is something to happen"	Interviews with user; discovery via model building

Table 4. Basic learning goals of a JCAT EUG, and sources we used to identify these goals.

Why is JCAT a valid target for an EUG?

JCAT has a number of properties that made it useful as a target for us to develop the EUG concept. First, it is primarily a modeling tool, which means that understanding *representation* will be critical for expert use. Next, users have often faced difficulty acquiring, understanding, and identifying the *data* needed to use the system. It also incorporates a number of underlying *algorithms* (Bayesian inference, event-based simulation) that are non-trivial and users may have difficulty grasping. Fourth, there are several aspects of information *display and visualization* that appear to present substantial obstacles for novice users. Thus, JCAT offers a glimpse into the four functions representative of intelligent software systems. In addition, we were able to work closely with the JCAT developer team, in order to understand their design choices, rationale, and the problems users have. Finally, there was a perception among potential users who had evaluated the system that JCAT was difficult to use, which indicates that something like an EUG could be useful.

Learning objectives for JCAT users

We derived learning objectives from a number of sources. First, we interviewed the development team, which gave important understanding of the goals and motivations of the system. Some of the developers were also experienced users of the tool, and so we were able to identify mis-steps and problems they had experienced. Some of the developers had also served as instructors or trainers for other users of the tool. This provided insight into the types of problems novice users of the system have. In addition, there is a small user community with a web-based forum for asking and answering questions about the tool. Analysis of these questions allowed us to infer some other questions faced by novice users. We had interviewed several individuals who had used the system more extensively and engaged in observations of users learning JCAT for the first time. We had access to models developed by novices after participating in a day-long workshop to learn the tool, as well as a set of models developed by more accomplished users and developers. Finally, one member of our team spent substantial time learning to use the system, documenting the errors, misconceptions, and tricks learned along the way.

By evaluating this information, we identified nearly 40 learning goals that we anticipated could form the basis of an EUG. Table 4 shows a basic description of these goals, along with the sources from which we learned about their importance. We have categorized these goals into the four main functions of intelligent software that we identified earlier. As expected for a modeling tool, we identified a large number of potential problems with how users understand representation. Also, we found a number of goals regarding underlying computations. For JCAT, these computations are fairly transparent, such as basic probability combination rules. Data generation and output display are less prominent in JCAT, although there are several issues related to these tool functions as well.

Vignette Development

Once learning objectives are identified, vignettes (of the type found in Table 3) can be designed to support the objectives. As an example, one problem novice JCAT users face is a misinterpreting the causal network, often because they are familiar with other ways network diagrams are used. These uses include PERT charts for project planning, social networks, concept maps, organization charts, influence diagrams, and so on. Mistaking a node in JCAT for one of the ways that nodes are interpreted in other tools can lead to misinterpretation of valid models, and can prevent a user from building a valid model. As an example, the conditional probability that one event occurs (“an engine fails”) given that a cause happened (“bad fuel was used”) is in theory and practice something that can be estimated. But if the nodes are thought of as entities in the system (“engine” and “fuel”), it becomes difficult to identify what the proper interpretation of a probability of “fuel” causing “engine”. A sample experiential vignette that might support this learning goal is shown in Figure 2. It takes the form of “troubleshoot model” scenario.

Demonstration Implementation of EUG and Future Steps

Our final EUG concept is still under development. We have made initial steps toward developing the software framework in which the EUG is embedded. Figure 3 shows concepts for this implementation. The left-hand screenshot shows an index screen, which allows a user to search for particular lessons or concepts, or follow a pre-specified plan through the vignettes. The right-hand screenshot shows a vignette in which the user is presented with a description of a situation and must choose between two models that best match the situation.

Because JCAT is a modeling tool, many of our vignette types are probably most relevant to the needs of other tools that focus on modeling. Future progress in developing the EUG will focus on several new software tools that rely more heavily on data and visualization, which are less critical for JCAT. We anticipate that this will help us identify new vignette types and understand additional ways in which users have misconceptions and problems using intelligent software tools.

Background:

A mechanic wants to understand the role that bad fuel played in his recent engine failure. He builds the following model using JCAT, because the fuel system feeds into the engine:

Next, he wants to estimate the probability on the link between the two nodes: How likely is bad fuel to cause his problem. His first thought was to estimate the probability that fuel will reach the engine if it is bad. He knows this is high—nearly 100%, because the engine won't fail if the fuel does not make it to the engine, so he rejects this idea. Next, he considers the probability that that he used bad fuel. He estimates this to be 80%, because the car's owner was very suspicious that some neighborhood kids put something in his gas tank as a prank. He enters this number into the model as the conditional probability between fuel and engine. Finally, he needs a “trigger” for the model, so he schedules the probability of fuel for 100% at time 0. He runs the model and suspects there is a problem. First, the probability of engine failure due to bad fuel is at .8, which seems too high. Also, he thinks “My technical service bulletin said that sugar in fuel will cause this type of engine failure only rarely (about 15% of the time; it is more likely to foul the fuel pump or fuel line), and I never entered that into the model”. Build a new model that addresses the mechanic's problems and computes the proper probability of engine failure.

Solution:

The mechanic's first mistake is in how he framed the model, with nodes called “fuel” and “engine”. In JCAT, nodes represent events: possible states of the world that occur probabilistically. Fuel and engine are entities, and not events. A better model would name these events appropriately, which will enable probabilities to be estimated better. These events might be called “Bad fuel was used”, and “Engine Fails”

Figure 2: Vignette for training the meaning of nodes and links in JCAT

Figure 3: Concept for an Experiential User Guide.

The guide is written in dynamic html, and provides a basic index screen (left) that allows individual vignette-based modules containing screenshots and text-based lessons to be selected (upper right).

CONCLUSIONS

This paper described our basic approach for developing a new type of user guide to support intelligent software tools. This *Experiential User Guide* attempts to expose a user to many of the experiences that an expert will have over a period of weeks or months of use that allows them to understand the strengths and weaknesses of the tool. The experiential training approach is useful for many learning objectives, but we believe it to be especially useful for learning intelligent software systems, in which textbook lessons may simply be inadequate to convey the complex reasoning functions the software tool is doing.

ACKNOWLEDGEMENTS

The research reported on here was performed under Subcontract No. TACS-SC-04-144, TO 0013 from Northrop Grumman IT (Prime contract FA8650-04-D-6546). We would also like to thank Dr. Janet Miller and Cheryl Batchelor with AFRL/RHCF and Robert Hoffman from IMHC for their encouragement and guidance.

REFERENCES

- Child, J., Bertholle, L., & Beck, S. (1961). *Mastering the Art of French Cooking*. New York: Knopf.
- Dornfest, R., Bausch, P., & Calishain, T. (2006). *Google Hacks*, 3rd ed. Sebastopol, CA: O'Reilly.
- Ead, J. (1996). *Chess for Dummies*. Foster City, CA: IDG Books.
- Evens, M., & Michael, J. (2006). *One-on-one Tutoring by Humans and Computers*. Mahwah, NJ: Lawrence Erlbaum.
- Hall, D. (2002) *The Starving Students' Cookbook. The Classic Guide for the Clueless Gourmet*. New York: Warner.
- Lemmer J. & McCrabb, B. (2005). A genuinely bayesian tool for managing uncertainty in plans. Available from <http://uncertainreasoning.com>
- MacDonald, M. (2006). *Excel for starters: The missing manual*. Sebastopol, CA: Pogue Press/O'Reilly.
- Musser, G. (2008). *The Complete Idiot's Guide to String Theory*. New York: Alpha Books.
- Nissenberg, S. K. (2002). *The Everything Kids' Cookbook: From Mac 'N Cheese to Double Chocolate Chip Cookies- All You Need to Have Some Finger Lickin' Fun*. Avon, MA: Adams Media.
- Rombauer, I. S. & Rombauer Becker, M. (1975) *The Joy of Cooking*. Indianapolis: Bobbs-Merrill.
- Singer, D., (1997). *Stage Writer's Handbook: A Complete Business Guide for Playwrights, Composers, Lyricists and Librettists*. New York: Theatre Communications group.
- Stevens, S. & Klarner, A. (1990). *Deadly Doses: A Writer's Guide to Poisons*. Cincinnati, OH: Writer's Digest Books.
- Tognazzini, B. (1998). Designing Single-Use Applications. <http://www.asktog.com/columns/010minaturegolf.html>.
- Tufte, E. R. (2001). *The Visual Display of Quantitative Information*, 2nd Edition, Cheshire, CT: Graphics Press