Steptorials: Mixed-Initiative Learning of High-Functionality Applications

Henry Lieberman
MIT Media Lab
Cambridge, MA, USA
lieber@media.mit.edu

Elizabeth Rosenzweig
Bentley University
Waltham, MA, USA
erosenzweig@bentley.edu

Christopher Fry
MIT Media Lab
Cambridge, MA, USA
cfry@media.mit.edu

ABSTRACT

How can a new user learn an unfamiliar application, especially if it is a *high-functionality (hi-fun)* application, like Photoshop, Excel, or programming language IDE? Many applications provide introductory videos, illustrative examples, and documentation on individual operations. Tests show, however, that novice users are likely to ignore the provided help, and try to learn by exploring the application first. In a hi-fun application, though, the user may lack understanding of the basic concepts of an application's operation, even though they were likely explained in the (ignored) documentation.

This paper introduces *steptorials* ("stepper tutorials"), a new interaction strategy for learning hi-fun applications. A steptorial aims to teach the user how to work through a simple, but nontrivial, example of using the application. Steptorials are unique because they allow varying the autonomy of the user at every step.

A steptorial has a control structure of a reversible programming language stepper. The user may choose, at any time, to be shown how to do a step, be guided through it, use the application interface without constraint, or to return to a previous step. It reduces the risk in either trying new operations yourself, or conversely, the risk of ceding control to the computer. It introduces a new paradigm of mixed-initiative learning of application interfaces.

Author Keywords
Steptorials; tutorials; help; documentation; program stepper;

ACM Classification Keywords
H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

IUI'14, February 24 - 27 2014, Haifa, Israel

ACM 978-1-4503-2184-6/14/02···$15.00.
http://dx.doi.org/10.1145/2557500.2557543

HI-FUN VS. LO-FUN APPLICATIONS

As the range of tasks that people want to do with computers expands, and the capability of software grows, we are faced with the development of *hi-fun* interfaces. We are not going to give a precise definition of hi-functionality interfaces here. Roughly, we mean those that provide large command sets, long menus, large or numerous icon bars, many data types, and complex patterns of use. *Low-functionality (lo-fun)* interfaces are much simpler, acting on just a few kinds of data, and providing reasonably small command sets, where the name and effect of each command are expected to be immediately apparent to the user.

Apple's *Preview* is an example of a relatively lo-fun application for images; it can, for example, print, crop, and rotate images, but it has relatively few operations (about 9 top-level operations, 7 menus of 5-15 items, few subsidiary dialogs). Adobe's *Photoshop* is a hi-fun image application (25 top level operations (+ modifier keys on many), 4 palettes of 2-3 tabs each, 8 menus of 10 to >25 items, many subsidiary dialogs). It has many different image types, and the total number of operations reaches into the thousands. It has a number of abstract concepts that it is necessary to learn, such as layers. It is user customizable, can record and play macros, has numerous plug-ins, etc.

The paper proceeds with a discussion of the problem of learning hi-fun interfaces, with identification of the ability to dynamically vary the autonomy of the user as a key. We then present *steptorials*, a new interaction strategy that allows this variation, and show steptorials implemented in the decision-support system Justify. We then report usability testing.

THE CHALLENGE OF LEARNING HI-FUN APPLICATIONS

Applications that become popular tend to grow into hi-fun interfaces over time as users desire more features and companies continually try to improve their products.

The most successful, like Photoshop or Microsoft Excel, become languages and programming environments in their own right. They become as powerful (and as difficult to learn for new users) as interactive development environments for programming languages.

The UI for hi-fun applications is typically designed for the expert, habitual user. It aims to make all the operations that the expert user would want to use easily accessible. But then the new user doesn't know where to start. And users who try to learn an interface by sequential exploration get confused because they are tempted to try many things for which they won't have use until much later, if at all.

THE PARADOX OF HELP

Developers of hi-fun interfaces are well aware that novice users may have trouble learning them. So they provide help, in a variety of forms. Applications may be introduced to new users with an introductory video, which shows an example of a typical use of the application. Tutorials may be presented, which guide the users step-by-step. The tutorials may take the form of a set of screenshots, with explanations of each one, or, more recently, various kinds of interactive tutorials may allow the user a more participatory role. Within the application itself, help may be provided with a help menu with index and search on particular topics. You may be able to point to a particular interface element and receive so-called "tooltips" by hovering the pointing device at that location. There are a wide variety of ways of offering help, and some complex applications essentially offer them all.

So why do users still have trouble? We know that people learn in both top-down and bottom-up ways [12] and for that reason some users may need to learn by exploring. User studies [9] reinforce this concept, but surprisingly, show that the majority of users are persistent in trying to use an application without first seeking help. This is especially true for novice users, and for complex applications, precisely the situation in which help would provide the most benefit. So what's wrong, and what do we do about it?

AUTONOMY, CONTEXT, RISK, AND STYLE IN LEARNING INTERFACES

We would like to focus on four major issues that affect whether users will be able to make effective use of help: *autonomy*, *context*, *risk,* and *cognitive style*.

Different forms of help provide different levels of *autonomy* to the user. At one extreme, watching a video affords the user no autonomy at all-they can only sit there, eat popcorn, and watch. If they get tired or bored, they can stop the video but that's all. Similarly, reading a manual is also passive. The reluctance to give up control is a principal reason why users are reluctant to view prepared material before attempting to use an interface.

At the other extreme, forms of help that are embedded in the application's interface provide the most autonomy. The user is free to use any part of the application they wish, and call up the help at any time and place they want. This is fine if the subject of the help is localized to a particular interface element, but is often unhelpful when the user has to consider patterns of use that might stretch across several operations or several interface elements. Between these extremes, interactive tutorials provide an intermediate level of autonomy, and are often preferable for that reason.

But it is hard to choose an appropriate level of autonomy that is right for all situations. Most people learn best by doing, that is, when they have a maximal level of autonomy. But novice users may get lost if left on their own, and it may be necessary to cede some autonomy to a more knowledgeable teacher for effective learning. Users may be more or less expert, or more or less confident in various aspects of an interface. The problem with traditional forms of help is that you have to choose a fixed level of autonomy at the start.

The second problem is *context*. Watching a video, reading a manual, or looking at help that appears in pop-up windows, all take you out of the context of using the actual application. In-place help, such as tool tips, and some forms of interactive tutorials, such as stencils [5], are much better at preserving relevant contexts of use. However, providing in-place help might be constrained by screen real estate or the disadvantage of obscuring portions of the interface.

The third problem is *risk*. When users cede autonomy to the computer, they take the risk that it will not be rewarded. The video may go too fast to get an effective understanding or too slow, boring the user. When the user launches a tutorial, they are often not sure how much time they are committing. The well has been poisoned by marketing-hype videos, or poorly written documentation, making users reluctant to turn to documentation for help. Much documentation and help is written in technical jargon.

The last problem we would like to consider is that of *cognitive style*. Users vary enormously in their preferred style of learning. Some people are top-down learners. They like to understand things conceptually before embarking on practical procedures. Others are bottom-up learners. They need to start exploring and doing first. Conventional help doesn't provide enough flexibility for learners of different cognitive styles.

VARIABLE AUTONOMY IN USER HELP

The solution we propose in this paper is to allow the user to *vary the autonomy level at every step in the interaction.* At every step of the way, users can choose whether they would like to sit back and have the computer present something automatically, whether they would like to do it themselves, or whether they choose some point in between, leading to a mixed-initiative interaction.

Not having to choose a fixed level of autonomy in advance means that interactions can be tailored to the level of expertise of the individual user for that particular part of the application, supporting different *cognitive styles*. Depending on the situation at the moment, the user can choose help either in or out of *context*.

Finally, having a variable level of autonomy reduces the *risk*, since the user can always go back and choose a different level of autonomy without any penalty.

LEARNING GOALS

The goal for introducing a hi-fun application should be to get a new user "up to speed" in the time they would plausibly allocate for a session with a new program; say 20 minutes to an hour at most. They should be able to gain the skills to complete a simple example that seems realistic to them. To motivate them to continue learning, they need to experience the "magic" of having the system be able to do something for them that they would have found difficult or impossible to do by paper-and-pencil, or other manual means.

After successful completion of the introductory example, they should also be able to understand what the paths are to learn more about the interface. It's usually not possible for them to learn any significant portion of an entire hi-fun interface in an introductory session, but they should get a sense that there is a world of functionality at their fingertips just waiting to be discovered.

INTRODUCING STEPTORIALS

This paper introduces the idea of a *steptorial* ("stepper tutorial"). A steptorial is a kind of interactive tutorial based on the control structure of a reversible programming language stepper.

The idea is that the interface steps necessary to complete the introductory example are like a "program" (described by English sentences and/or interaction with the application rather than programming language code). The steptorial allows the user to step through the example, as a programmer steps through code. The steptorial is completely reversible, inspired by the control structure of the program stepper ZStep [6]. In extending the stepper metaphor beyond its origins in program debugging, we are enabling learning by *end-user debugging* [7] of application use-cases.

THE CONTROL STRUCTURE OF A STEPTORIAL

The control structure of a steptorial follows the control structure of a program stepper. Steppers traditionally display the code for a program, with a program counter indicating the expression that is just about to be executed.

The user has the choice of *Stepping Over* the current expression, which executes it, returns the result, and moves the program counter forward to point to the next expression. The user also has the choice of *Stepping Into* the current expression, which dives down into the details of evaluating the expression, which, recursively, are stepped. The *Step Over/Step Into* choice represents a mechanism for control over the level of detail. In debugging, this prevents the programmer from getting drowned in detail while preserving the ability to see any particular detail if the need arises.

The fully reversible ZStep debugger [6] introduced the insight that it's often hard to make the choice about whether or not you want to see detail of an expression until *after* you see its result. If you choose not to see detail, and it turns out an error did in fact occur in execution, you can back up the stepper, and only then go through the detail of what may have caused it. Though there have been several reversible stepper implementations, none of today's most popular programming language IDEs come with a reversible stepper as standard equipment.

We aim to bring the same kind of flexibility to choices about how much autonomy and risk the user wants to take while following an introductory tutorial. The steps of a steptorial are represented in English sentences rather than the program code found in a stepper. Both represent, in some sense "instructions".

At each step of the tutorial, the user is given choices that vary in autonomy, from "have the computer do it" to "let me do it myself". Since the steptorial is reversible, the user can always back up and make a different decision. A good learning strategy for top-down learners is often to passively watch a step at first, then increase autonomy by trying it themselves. For bottom-up learners, they might want to try a step first, then retreat to more guided modes if they run into problems.

AUTHORING STEPTORIALS

At the moment, steptorials are hand-authored for each example. Each example is described as a set of steps, each roughly representing a user interaction which satisfies a single goal. The step is represented to the user by an English sentence describing the goal. Each step describes the interface operations necessary to accomplish it. Steps may have substeps, each of which corresponds to a subgoal. Justify has an interpreter for running a steptorial from the list-based representation that contains the steptorial description as above.

Future work will consider the possibility of generating steptorials automatically or semi-automatically from program demonstrations or from user-supplied examples.

STEPTORIALS IN JUSTIFY

The remainder of this paper will present a steptorial for the decision-support application *Justify* [4]. Justify is a hi-fun application that helps manage structured online discussions about important issues. Justify has a total of 4808 interface operations, making it comparable to Photoshop (whose documentation index contains 4032 entries, almost all of which describe a particular tool or interaction).

Justify is organized around threaded discussions composed of *points* (like *posts* in an online forum). Each point represents a single idea, fact, or opinion. Points have a rich ontology of types, which represent the role of that point in the argument, such as *pro* or *con*. Each point also has an *assessment*, the result of applying a rule (based on the type of point) that is intended to compute a summary or possible decision for that point, taking into account the points below it.

Like a spreadsheet, creating, deleting, or changing points propagates changes automatically to other points that depend on it. Justify is actually a *language* for representing arguments, and the interface is really an interactive development environment (IDE) for that language [4], which is what makes it hi-fun.

CHOICES IN THE JUSTIFY STEPTORIAL

The Justify steptorial window (Figure 1) provides, as we mentioned earlier, the standard stepper controls: Forward and back one step or to the beginning or end (arrows along left side), and the *"Dive In"* icon, which steps *into* the current step, revealing its details. For each step, we provide three choices (along bottom): *Show Me How to Do It*, *Guide Me Through It*, and *Let Me Do It Myself*. The purple right-pointing arrow is the "program counter" indicating the current step.

If the user chooses *Show Me How to Do It*, it runs a predefined video that corresponds to the current step. If the user does *Step Over* (down arrow) on that step, it runs the entire step to completion. If the step happens to have substeps, videos for *all* of the substeps are concatenated and run continuously.

If the user instead chooses *Dive In* for that step, the substeps for that step are opened up, and the first substep is indicated. In this way, the user may choose any of the three presentation options for each substep independently. When all the substeps for the main step are finished, control returns up to the previous level, following the control structure of a standard stepper.

Future versions of the steptorial will replace the video segments with execution of the procedure in the actual application itself (in its own window and affecting its own internal data structures). This requires the application to be externally scriptable. Reversing the stepper will fully restore application state.

Guide Me Through It launches a *stencil* based interactive tour [5]. Stencils "gray out" all interface elements except the one with which the user is expected to interact at that particular step. Stencils permit interaction in the proper application context, and direct the user's attention to avoid distraction. Again, the level of detail seen is determined by the *Step Over/Step Into* distinction. If we step into a substep, we play only that part of the stencil tour that is relevant to that substep.

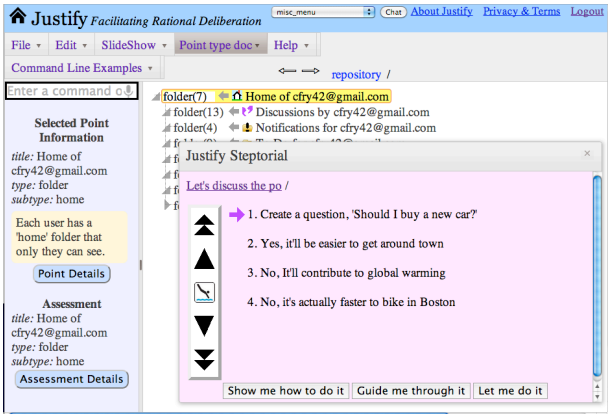

Figure 1. A *steptorial* (lower right pane) in the decision support system Justify.

Stencil tours can vary in the level of interaction they permit the user. Allowing the user more autonomy in operating the interface means that the application has to check to make sure the user isn't getting "off track".

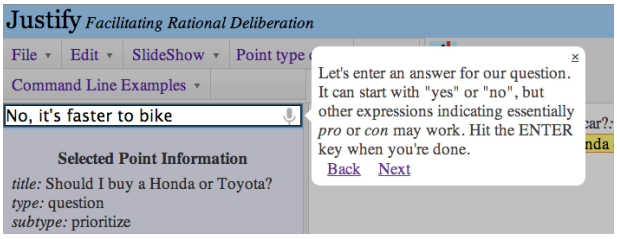

Figure 2. A *stencil* guided tour. Most of the interface is "grayed out", except for the particular element that the tour wants the user to interact with at that step.

Finally, there's *Let Me Do It Myself*. This allows the user pretty much unconstrained use of the application, with the understanding that they will try to accomplish the goal represented by that particular step themselves. The stepper controls are replaced by two options: *I Did It*, and *Oops!*,

indicating, respectively, the user declaring success or failure of the goal.

If the application is scriptable and recordable, it may be able to determine for itself whether the user accomplished the goal, rather than wait for the user to click *I Did It*. This may obviate the need for the user to explicitly mark the end of the interaction.

Oops! simply returns to the previous application state, so there's little risk for the user in at least attempting to perform the task themselves. There's also potential, as in Intelligent Tutoring Systems, for an intelligent program to try to diagnose what may have gone wrong with the user's attempt, and try to provide more targeted help.

EVALUATION

First, we ran a usability test on the Justify system itself. This study had 6 participants, over an hour each, and focused on the introductory experience. Justify provided help in a variety of conventional forms: video, side-panel help, and tool tip help on interface elements. Though this study is not the subject of this paper, we uncovered problems typical of introducing hi-fun interfaces to new users: disorientation, distraction, and unwillingness to consult the provided help. This motivated the introduction of steptorials.

We then ran two kinds of formative evaluation on Justify steptorials. The first was a Heuristic Review [8] performed by two experienced User Experience (UX) professionals. We also performed a walkthrough with a single novice user, experienced with Microsoft Office and other complex applications, but not a programmer. We were most interested in the question of whether a steptorial would support the preferred learning style of the user, regardless of whether it was top-down or bottom-up.

Heuristic review

The two UX reviewers represented two contrasting learning styles: one preferred a learning style that starts with exploration; the other preferred to read or view explanatory material before exploration. Both found that the Steptorial interaction allowed them to learn about the interface using their preferred style. Each felt that they were more "engaged" with the interface than with conventional help systems, and that the interface provided more "motivation for pushing through complexities in the interface".

User walkthrough

The novice user walkthrough, (and user testing of new help paradigms in general) provides some interesting challenges. Most users' experience with conventional help is so discouraging, that if we merely offer a steptorial as an alternative to traditional forms of documentation, we run the risk that they will ignore steptorials in the same way they've been shown to decline other forms of help.

This test was *not* designed to see if users could pick up the concept of a steptorial from scratch and spontaneously choose it over conventional help. Therefore, the session started out with an introduction to the steptorial methodology, and we verified that the participant understood the operation of the steptorial window. We were interested to see whether the participant could achieve a better understanding of Justify than was possible with conventional help methods.

Participant A. was able to learn the steptorial methodology easily after the initial presentation. She then embarked on learning Justify. The task was to encode a simple argument, with one yes-or-no question, one reason in favor, one reason opposed, and a reply to refute one of the first two reasons presented.

There were some problems with orientation and context. A. had some trouble distinguishing real-world or problem-focused concepts from interface concepts. For example, a *question* refers both to a (real-world) topic the user wishes to discuss using Justify, and a technical concept in the Justify interface that the steptorial is trying to teach you how to create. A. sometimes got confused about what constituted a "step". After introducing the question, the intent of the steptorial was to make three sequential entries, each professing an opinion for or against the question. But A. at first interpreted them as three different options from which she should select just one. Writing steptorials is an art, just like writing other kinds of documentation, and care is needed to keep terminology straight.

Because of the "low-fidelity" of the prototype (all modes were not fully implemented), A. sometimes got out of sync between the various modes. Running a video looks almost identical to actions in the interface, but when you return to the interface, the effects of the actions aren't there! Similarly, we were unable in the current implementation to keep the stencil guided tour in perfect sync with the stepper navigation. We hope to correct that in future when we are able to fully script the application.

Before the test, we had asked participant A. about how she normally approached learning new applications. She said that she preferred to "dive right in" rather than preparing by reading manuals or watching videos. However, we observed that during the test, whenever she was presented with a choice between watching the video, guided tour, and unconstrained application use, she chose to watch the video first. But it also shows that, even though a user might have a preferred overall strategy, they may choose in the moment to adopt a different strategy for a specific situation. This confirms our hypothesis that it is valuable to offer varying autonomy alternatives at every step.

At the end, A. demonstrated that she was able to operate the interface to enter her own example with little or no help, analogous to the example she was shown. We take this as a positive result, especially as the previous tests had shown

that users who did not follow tutorials had trouble with similar problems.

RELATED WORK

The most closely related work is Chi et. al's MixT, [2] which provided the user a choice between static (screenshot+text) tutorials and video tutorials. In both cases, the tutorials were presented linearly, rather than with the structure of the task hierarchy, as here, and there is no reversibility.

We employ Kelleher and Pausch's stencil [5] technique as an intermediary between fully scripted and fully flexible options for the user. Selker's Coach [10] system pioneered the use of user-generated examples and mixed-initiative tutorials in teaching rather than predefined tutorial material. We see opportunity in incorporating more user-generated examples and input into steptorials.

Carroll and Rosson [1] and Fischer [3] have long advocated for users taking an active role in learning interfaces. We have been inspired by their user-centered design principles, relevance to user goals over techno-centrism, and advocacy of learning by doing.

We see this work in the tradition of Intelligent Tutoring Systems [11] and many opportunities arise for trying to better understand user behavior and provide personalized help. VanLehn [13] surveys contemporary ITS'es and compares with human instruction. Wiedenbeck and Zila [14] systematically tested guided vs. exploratory approaches.

CONCLUSION

Hi-functionality applications are here to stay. If we're going to enable new users to be productive quickly, we need better ways of introducing users to them. They need to be able to quickly succeed at small, but nontrivial and relevant examples that best show off the use of the application, to motivate them to continue learning and using it. But we can't teach them everything at once.

This paper introduced the *steptorial*, a new paradigm for mixed-initiative learning of complex applications. It allows users to choose at any moment, between passively watching a demonstration of that step, trying it themselves without help, or via one or more mixed-initiative learning modes. Whether you prefer a top-down learning style, or a bottom-up learning style, we'll get you on the road to success.

ACKNOWLEDGMENTS

We would like to acknowledge Dani Nordin and Heather Wright Karlson for their help with the usability evaluation.

REFERENCES

1. Carroll, J., Rosson, M. B., The Paradox of the Active User, in *Intefacing Thought: Cognitive Aspects of Human-Computer Interaction*, MIT Press, 1987.
2. Chi, P., Ahn, S., Ren, A., Dontcheva, M., Li, W., Hartmann, B., MixT: Automatic Generation of Step-by-Step Mixed Media Tutorials, ACM User Interface Software Technology (UIST), 2012.
3. Fischer, G., Lemke, A. C., & Schwab, T. (1985) Knowledge-Based Help Systems in L. Borman, & B. Curtis (Eds.), CHI 1985, pp. 161-167.
4. Fry, C., Lieberman, H., Decision-Making Should Be More Like Programming, Int'l Symposium on End-User Development, Copenhagen, June 2013.
5. Kelleher, C. and Pausch, R., Stencil Based Tutorials: Design and Evaluation, CHI 2005,pp 541-550
6. Lieberman, H. and Fry, C., ZStep 95: A Reversible, Animated, Source Code Stepper, in *Software Visualization*, John Stasko, John Domingue, Marc Brown, and Blaine Price, eds., MIT Press, 1997.
7. Lieberman, H., and Wagner, E., End-User Debugging for Electronic Commerce, Int'l Conference on Intelligent User Interfaces, 2003.
8. Nielson, J, Technology Transfer of Heuristic Evaluation and Usability Inspection, IFIP INTERACT'95 International Conference on Human-Computer Interaction, Lillehammer, Norway, 1995.
9. Pashler, H, McDaniel, M, Rohrer, D, Bjork, R., Learning styles: Concepts and Evidence, Psychological Science in the Public Interest, 9, pp 105-119, 2008.
10. Selker, T., Coach: A Teaching Agent that Learns, Communications of the ACM, July 1999, Vol.37 No.7, pp. 92-99.
11. Sleeman, D. and Brown, J.S., *Intelligent Tutoring Systems*, Academic Press, 1982.
12. Sun, R., Zhang, Xi, Top-down versus bottom-up learning in cognitive skill acquisition, *Cognitive Systems Research* Vol. 5, pp. 63-89, 2004.
13. Van Lehn, K., The Relative Effectiveness of Human Tutoring, Intelligent Tutoring Systems, and Other Tutoring Systems, Educational Psychologist 46(4), 197-221.
14. Wiedenbeck, S. & Zila, P. L. Hands-on practice in learning to use software: a comparison of exercise, exploration, and combined formats, ACM Transactions on Computer-Human Interaction, Vol. 4., No. 2, pp. 169-196, 1997.