Discoverable Computer Interfaces to Teach Users Efficient Controls

Aaron Moss (3217641)
moss.aaron@unb.ca

3 April 2011

Supervised by Dr. Cosmin Munteanu & Dr. Michael Fleming

Abstract

This paper contains an investigation of how different kinds of computer interface controls facilitate users' discovery and learning of the functionality afforded by these controls. To test these capabilities, I have developed a simple rich text editor with both a standard graphical interface and a more efficient keyboard-based interface which uses MediaWiki formatting markup. This study examined the utility of providing a syntax highlighting-based "live preview" of the formatted text for user efficiency, satisfaction, and education in the use of the more efficient typed formatting markup.

Analysis of the data collected from the study suggests that the live preview feature is effective at improving user efficiency, but through reducing user invocation of the manual preview feature rather than by training users to utilize keyboard-entered formatting markup. In fact, the greater similarity of the live preview-enabled interface to common word processing software appeared to suggest to its users a greater similarity of behaviour as well. The experimental group of users, who were given a live preview-enabled interface, in fact showed less aptitude than a control group without the feature to discover the usage and functionality of the typed formatting markup.

Contents

1 Introduction **3**

2 Background **3**
2.1 Multi-Layered Design 4
2.2 Relative Efficiency of Keyboard and Mouse-Based Interfaces 5
2.3 Markup and Syntax Highlighting 6

3 Research Outline **7**

4 Methodology **7**

5 Results **10**
5.1 Initial Assumptions 10
5.2 Markup Discovery and Use 11
5.3 Use of Preview Feature 12
5.4 Error Rates 13

6 Conclusion and Future Directions **13**

References **15**

A Glossary **16**

B User Survey **17**

List of Figures

1 Multi-layered design 5
2 LiveWiki screenshot 8

1 Introduction

Computer user interfaces that are easy to learn how to use often do not afford the same functionality as interfaces that are unintuitive and which require more time to learn how to use [11]. In this report, I have investigated a method by which a computer program can train its users in a more efficient method of control. This paper will detail the methodology and results of my research, after first providing some background information on the topic.

2 Background

In the field of human-computer interaction (HCI), there is often conflict between the design goals of discoverability and efficiency [11]. Many interfaces that have been designed to be highly efficient to use do not afford an obvious method of use, while interfaces that have been designed to be discoverable and to naturally suggest their usage may require even advanced users to perform many actions to accomplish a single task. GNU Emacs[5] and LibreOffice Writer[6], two programs designed for authoring textual documents, demonstrate these contrasting design philosophies — Emacs has many powerful but complicated keyboard shortcuts, sometimes requiring four-key combinations, while Writer exposes most of its features through menus and dialog boxes, requiring multiple mouse clicks for some commands. Both inefficient and non-discoverable user interfaces contribute to loss of productivity and public frustration with computer technology, the former by inconvenience to advanced users, the latter through learning challenges to novice users. [11]

This conflict arises in that computer programs are typically designed to support both novice and expert frequent users. However, the user interface qualities demanded by both these groups are often at odds, forcing designers to make trade offs that serve neither group optimally. Novice users often desire detailed help and clear, step-by-step guidance to perform their tasks, requiring interface designers to break tasks down into smaller units, and provide more frequent and detailed feedback and instruction. Expert users, on the other hand, are

deeply familiar with the system and its interface, and require significantly less feedback and instruction; these users are in fact hindered in their work by the program informing them of what they already know, and desire the system to unobtrusively and non-distractingly facilitate the efficient completion of their workflow. These frequent users also often want to be able to compose frequently performed sequences of actions into efficiently runnable "macros", in contrast to the desire of novice users for tasks to be presented in smaller chunks. Obviously it is difficult to meet such disparate goals in the same user interface. [11]

2.1 Multi-Layered Design

One approach that has been proposed to resolve the conflict between the requirements of novice and expert users is multi-layered design [12]. In a multi-layered design, interface functionality is divided into "layers", each layer containing a set of complexity, and each subsequent layer adding to the complexity of the layers before it (see Figure 1). This complexity may be system functionality, increased data, or new levels of abstraction [2]. Clark and Matthews[3] have further classified layers as either "feature" or "mixed", where a feature layer contains a set of related functionality, while a mixed layer is organized solely by complexity, and may include several otherwise unrelated functions.

While a layered design has been shown to be effective at introducing novice users to complex systems [2, 9], multi-layered design also has a number of disadvantages stemming from its use of multiple versions of the system interface. One of these disadvantages is that the layering system itself is an additional complex feature that the user must learn; this additional complexity may outweigh any simplifying benefits provided by the layering system [2]. Related problems are the difficulty of designing training for when the user should switch layers [12], and the difficulty users have learning a new layer if it adds too much new functionality [2]. A further problem in multi-layered design is that it is difficult to separate program functionality into distinct, sequential layers [12]. Users may be at an overall low level of sophistication with regards to the system, yet require a few features that the interface

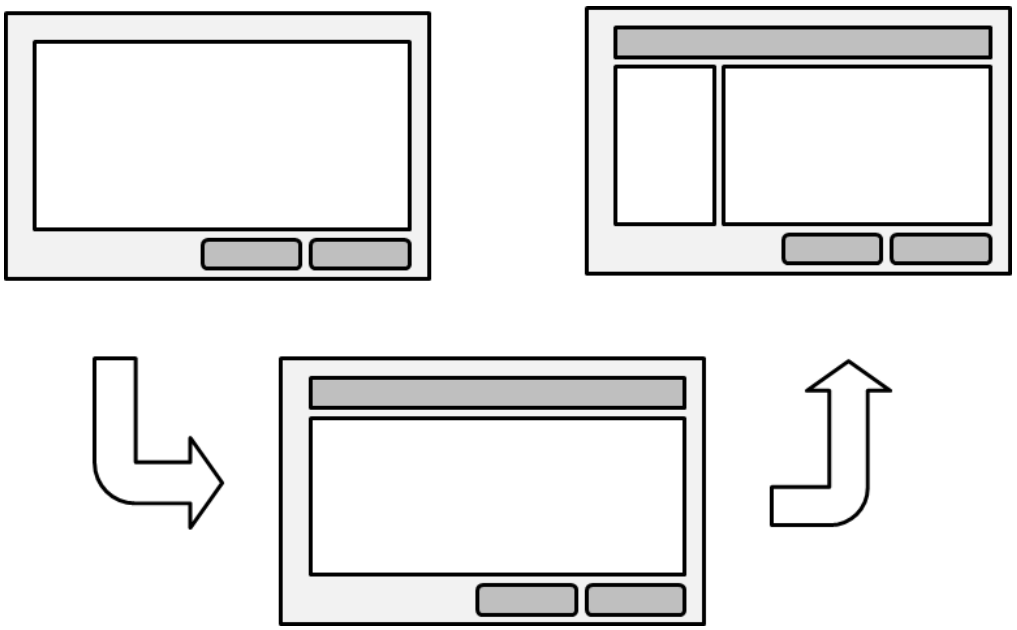

Figure 1: An illustration of different layers in a multi-layered design.

designer has placed in an advanced layer, and thus are forced to use the advanced layer, negating the utility of the layered design. While the concept of feature layers proposed by Clark and Matthews [3] largely mitigates this problem, the fact remains that different users will have need of different subsets of a system's functionality, and these subsets may not be able to be grouped clearly into any sort of layered design. Finally, advanced users often see no need for the added complexity of multi-layer design, as they are satisfied with an interface which exposes all its features at once [2].

2.2 Relative Efficiency of Keyboard and Mouse-Based Interfaces

Some of the benefits of layered design can be attained by use of multiple input methods for a homogeneous interface. While not providing all the simplicity of the more basic layers of a multi-layered design, using multiple input methods does have the benefit of consistency. For example, a single interface may have both keyboard and mouse-activated controls. As shown by the comparison of Emacs to Writer, keyboard-driven interfaces are often more efficient, while mouse-based interfaces tend to be more discoverable.

For tasks primarily involving textual input, a keyboard-based interface is the fastest and most efficient to use; if the user does not frequently need to select distant interface elements, the time required to move their hands and acquire another input device (*e.g.*, a mouse) outweighs the time saved from the mouse's quicker selection of distant elements [11]. Nonetheless, a graphical user interface (GUI) is more natural to novice users, and is typically manipulated with a pointing device such as a mouse. Unlike a keyboard-driven interface, which often requires its users to remember the textual commands to complete any action they wish to perform, GUIs display the necessary commands on the user's display, allowing the user to recognize the command, rather than recall it. [1, 11] A typical example of this would be to compare a graphical shell to its command-line equivalent. The graphical shell will display icons for each application available to be launched, while to operate a command-line shell, the user must remember and type the name of the application they wish to launch; it may be quicker to type the command than to find and select the icon on a crowded display, though.

2.3 Markup and Syntax Highlighting

Recognizing the efficiency of keyboard-based interfaces for textual input tasks, interface designers have attempted to use the keyboard to facilitate the entry of non-textual data in an equally efficient manner. One method to accomplish this is the use of "markup", additional text included in a body of natural text that provides extra information about that body of text [8]. A common example of markup would be the HTML markup used to format webpages; in this markup, a paragraph of text is surrounded by opening and closing paragraph tags, `<p>` and `</p>`, which are used to control the display of the paragraph, but not shown to the user.

Another user interface technique that is often used in conjunction with textual markup is "syntax highlighting", automatically formatting certain portions of text to emphasize significant features [15]. For instance, in the HTML example above, a syntax highlighting editor

would colour the `<p>` and `</p>` tags in one colour, and the paragraph text contained within them in another. Syntax highlighting is used to make the distinction between document text and its markup obvious to the user of the program, possibly increasing the discoverability of the efficient keyboard-based interface.

3 Research Outline

I investigated a user interface approach which draws on the benefits of markup and syntax highlighting to mitigate the effects of the conflict between discoverability and efficiency in computer interfaces. I hypothesized that if discoverable user interface elements could demonstrate to users how to use more efficient methods of control as a natural aspect of the normal operation of these discoverable elements, frequent and sophisticated users of the system would be trained to use the more efficient interface, while retaining the easily discoverable and familiar interface for more occasional and less experienced users. Under this assumption, users' usage of the more efficient interface mechanisms should be roughly proportional to their level of experience with the system, and thus their desire for greater efficiency and lack of need for discoverability.

4 Methodology

To test the validity of this approach to human-computer interaction, I developed a simple rich text editor, which I named "LiveWiki", modelled after Wikipedia's[13] MediaWiki[10] editor. A screenshot of LiveWiki is shown in Figure 2. LiveWiki allows text to be entered and formatted as either bold or italic, and includes two parallel interfaces to format text. The first, more easily discoverable interface consists of a toolbar with graphical buttons for bold and italic; the buttons have icons showing boldface and italic text, **B** and *I*, respectively. The second way to format text is by surrounding it with MediaWiki formatting markup — for instance, to italicize text, surround it with two single quotes, `''like this''`. This

second interface is entirely keyboard-driven, and, as such, should be more efficient. It is not, however, as discoverable as the first, GUI-based interface.

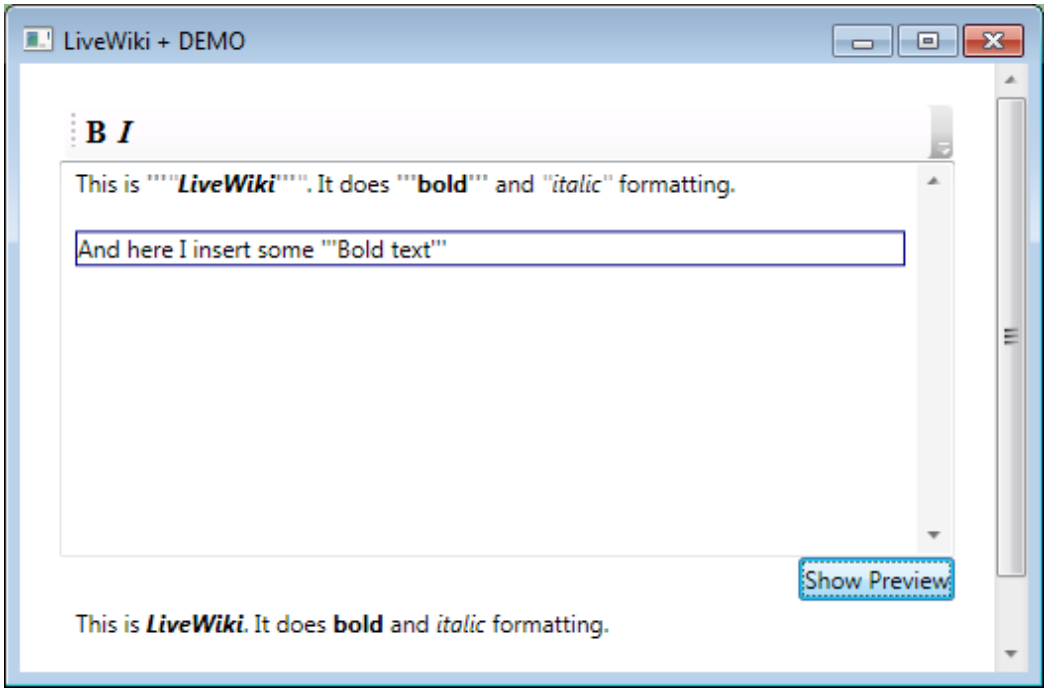

Figure 2: A demo of the LiveWiki software.

As in the Wikipedia editor, pressing the GUI buttons in my test program inserts the formatting markup into the text entry field. The user may then observe the results of this action by pressing the "Show Preview" button below this field, which will parse the text and markup in the text entry field, and show the formatted text below. LiveWiki operates in two modes — in one, its default mode, it simply emulates the Wikipedia editor, and the text entry field is entirely plain text, with no formatting, while in the second "live preview" mode, text in any paragraph the user is not currently editing is formatted much as it is in the preview. In the live preview mode, boldface text is shown in bold, italicized text in italics, and formatting markup is shown in grey, as well as being formatted appropriately. This preview is intended to give the users of the program more immediate feedback on the results of their actions, as well as demonstrating in a more obvious fashion the effects of the formatting markup, making the text entry field behave in a more "what you see is what you

get" (WYSIWYG) manner. I expected that this behaviour would suggest to users the use of the keyboard to enter formatting markup, rather than the graphical buttons, especially after repeated exposure, and that experienced users would prefer use of the keyboard to the GUI to format text.

To test this hypothesis, I recruited a group of test users ($n = 9$) from the population of students at the local universities. University students were chosen as the sample population due to availability, as well as the fact that students are familiar with computer programs, as necessitated by the technology used in a modern university teaching environment, but not generally experts in the use of markup-based editing tools, and thus a fairly representative sample of the computer-using public.

These users were shown a short demo of the LiveWiki editor's functionality, then asked to reproduce as much of a short document as they could in 20 minutes, using LiveWiki. The document was derived from the Wikipedia article on the potato[14], edited and reformatted to suit the purposes of this project. A Wikipedia article was used to provide a realistic set of test data for a wiki-style rich text editor. During this time, the editor kept logs of all user actions to determine metrics such as typing speed, number of errors, and use of keyboard or graphical controls. At the end of the 20 minutes, the test users were given a short questionnaire to determine their level of experience with related technologies, such as computers, word processors, and other types of formatting markup, as well as soliciting their subjective impressions of the test program in qualities such as efficiency, ease of use, and difficulty to learn (see Appendix B for the survey).

4 of the test users were given LiveWiki in its default mode, while 4 were given the program in live preview mode (one of these users is not included in the general statistics, due to a technical error). The remaining user was a pilot tester, and is not included in the statistics, but provided some interesting data, detailed below. The users testing the program in default mode were considered a control group, to determine the baseline behaviour of users exposed to a Wikipedia-style rich text editing interface, while the group testing in live preview was an

experimental group, to determine the benefits, if any, of such previewing functionality. This between-subject experimental design was chosen to avoid learning effects on the data of a within-subject experimental design, as if individual users tested both interfaces in sequence it would be difficult to discern what part of their markup comprehension was due to the second interface tested, and what part due to learning effects from the use of the first interface. My hypothesis was that the live preview group would show fewer errors, increased speed (through more extensive use of the keyboard to enter formatting markup), and higher subjective satisfaction than the control group, though my findings are somewhat weakened by the small sample size, as between-subject tests work better with a larger number of participants.

5 Results

After running the test and analyzing the resulting data, the experimental group does appear to have higher subjective satisfaction, fewer errors, and faster typing speeds, though the apparent reasons for these results were rather unexpected.

5.1 Initial Assumptions

One particularly interesting case was that of my pilot user. This user was given an earlier version of the program than I described above, and was not able to complete the test due to a program crash. In this earlier version, the live preview was applied to the formatting markup as the user typed, rather than after they left a paragraph. The pilot user's behaviour was to simply press the appropriate toolbar button until the text appeared bolded or italicized, ignoring the markup. When interviewed later, the user revealed a lack of understanding of the significance of the formatting markup, and an expectation that the program would behave similarly to Microsoft Word[4], a familiar application.

Based on the outcome of this test, the LiveWiki behaviour was modified to that detailed

above. This pilot test suggested that the live formatting preview would induce users to think that the LiveWiki editor behaved the same as common WYSIWYG word processors (such as Microsoft Word), an invalid assumption. The new behaviour, in which all text the user is currently editing is kept unformatted, while paragraphs the user is not editing are shown in live preview, was designed to oppose this mistaken assumption, while still maintaining many of the benefits of live preview.

As an additional response to this test, a blue box was drawn around paragraphs currently being edited in both versions of the program. This box was meant to visually differentiate the LiveWiki editor from common word processors, and to suggest an unformatted text entry field, following the principle that controls which appear similar will be expected to behave in a similar manner [7]. A further benefit is that the border clearly delineates the unformatted text from the formatted preview in the program's live preview mode.

Presumably due to these modifications, I did not observe the same behaviour of ignoring markup in any of the subsequent experimental group testers; these testers also showed much less frustration with the program interface than the pilot tester. While another study would be required to determine the benefits or detriments of character-by-character live preview relative to the paragraph-by-paragraph version I tested, I believe that keeping users' immediate text-entry area in plain text is beneficial to their comprehension of formatting markup.

5.2 Markup Discovery and Use

Even with the modifications to the test software, user familiarity with WYSIWYG word processors still had an effect on which of them discovered and used the formatting markup. 2 of the 4 control group users discovered the purpose and function of the typed markup, though one of these two did not actually type it; only 1 of the 3 experimental group users discovered the markup. The three users who discovered the markup gave the test program high ratings for ease of remembering keyboard commands, suggesting that these users found

the software an effective trainer. Furthermore, the user from the live preview group showed an increasing preference to type the markup once they discovered it, rather than enter it using the GUI controls. On the other hand, the control group user showed decreasing frequency of typed markup usage after an initial peak, perhaps due to uncertainty about its proper usage.

I believe the discrepancy in markup discovery between the control and experimental groups can be explained by the greater similarity of the appearance of the live preview interface to a word processor. All the users in my sample had at least 6 years' experience with Microsoft Word or an equivalent word processor, and many of them commented on the test software's lack of certain features found in that application, such as standard keyboard shortcuts (*e.g.* `Ctrl-B`, `Ctrl-I`) and spellcheck. Presented with an apparently familiar interface, the users in the live preview group did not investigate the anomaly of the formatting markup, but assumed that the program would behave like the word processors they had used.

5.3 Use of Preview Feature

One efficiency I observed in the experimental group was less use of the manual preview feature. From this I infer that the provision of the live preview functionality provided sufficient feedback on the correctness of the formatting input, and thus the completely WYSIWYG preview was unnecessary. The users in the control group used the manual preview feature an average of 4.75 times ($\sigma = 4.02$), while the users in the experimental group used the preview feature an average of 3.33 times ($\sigma = 2.87$). Preview usage proved to be an important factor in typing speed, with users showing higher words-per-minute rates on paragraphs where they used fewer manual previews (the Pearson correlation coefficient of per-paragraph words per minute to preview usage for the group that used the preview feature was -0.198). The users who actually discovered and typed the formatting markup took 3–4 uses of the manual preview feature to verify the markup's usage and functionality, using the preview in quick succession after small formatting edits. However, after this verification, these users showed

a noticeable decrease in usage of the manual preview feature.

5.4 Error Rates

The most significant difference between the control and experimental groups was the number of errors in their final documents. The experimental group averaged 0.06 formatting errors (misapplied or omitted formatting) per 100 words ($\sigma = 0.08$), as opposed to the control group's rate of 0.18 formatting errors per 100 words ($\sigma = 0.22$). The experimental group also had a lower average rate of transcription errors (misspelled, omitted, added, or substituted words) than the control group, 7.33 per 100 words ($\sigma = 2.49$) versus 13.25 per 100 words ($\sigma = 5.45$). The hypotheses that the experimental group has a lower rate of formatting errors than the control group and that it has a lower rate of transcription errors are supported at the 75% and 90% confidence levels, respectively (p-values 0.229 and 0.085), which is quite suggestive given the small sample size. Comparing the group of three users who discovered the formatting markup to the rest of the sample revealed similar trends, though with lower levels of confidence.

This data shows the benefits of live preview in reducing formatting error rates, and suggests that understanding of the formatting markup yields similar benefits. In the live preview case, this lower error rate is likely attributable to the greater fidelity of the program display to the source document, and thus greater ease of visual identification of errors. For the markup discovery group, the lower rate of formatting errors suggests that users are capable of accurately and efficiently parsing simple textual markup to determine its effects on a source document.

6 Conclusion and Future Directions

The problem of making computer interfaces both discoverable and efficient remains difficult to solve. The approach I pursued of providing a live preview of formatting markup showed

definite advantages, but did not seem to be effective as a training tool. Users given a wiki-style rich text editor with live preview enabled demonstrated increased satisfaction, as well as increased efficiency through reduced usage of the manual preview feature, but seemed somewhat less apt than a control group supplied with a plain text only editor to discern the functionality of the formatting markup. I attribute this primarily to prior experience with WYSIWYG word processors such as Microsoft Word, as the greater similarity of appearance of the live preview-enabled editor to a word processor suggests a greater similarity of behaviour than actually exists.

This work is exploratory in nature, and suggests multiple possible more detailed future investigations. One such investigation would be to compare the control functionality to a more traditional syntax highlighting behaviour, recolouring the markup without changing the formatting of the enclosed text. Given the results of this study, I expect this syntax highlighting would increase discovery and comprehension of the markup by making it more obviously differentiated from the surrounding text. Another avenue of investigation would be to make the paragraphs formatted by live preview fully WYSIWYG, hiding the formatting markup. Though this would reduce the potential usage of the formatting markup, it may also reduce usage of the manual preview further, resulting in efficiency gains. Modifying the test to include more complex tasks would also be interesting, as the simplicity of the transcription task may have hidden differences between the experimental and control groups. Possible such more complex tasks include modification of an existing document, or location and correction of formatting errors in markup. Including markup to define additional features such as headings, hyperlinks, or embedded images would also make the tasks more complex. Finally, a study conducted over a longer time period with a greater number of users would provide more statistically significant data, as well as a more in-depth investigation of the learning effects resulting from use of the system.

References

[1] Gregory D. Abowd, Elizabeth D. Mynatt, and Tom Rodden. The human experience. *IEEE Journal of Pervasive Computing*, pages 48–57, January–March 2002.

[2] Linn Gustavsson Christiernin. Multi-layered design — theoretical framework and the method in practise. In *Winter Meeting 2005 Proceedings, Department of Computing Science, Chalmers University of Technology*, 2005.

[3] Bryan Clark and Jeanna Matthews. Deciding layers: Adaptive composition of layers in a multi-layer user interface. In *Proceedings of 11th International Conference on Human-Computer Interaction*, volume 7, 2005.

[4] Microsoft Corporation. *Microsoft Word*. http://office.microsoft.com/en-ca/word/.

[5] Free Software Foundation. *GNU Emacs*. http://www.gnu.org/software/emacs/.

[6] The Document Foundation. *LibreOffice Writer*. http://www.libreoffice.org/features/writer/.

[7] Don Gentner and Jakob Nielsen. The anti-mac interface. *Commun. ACM*, 39:70–82, August 1996.

[8] C. F. Goldfarb. A generalized approach to document markup. In *Proceedings of the ACM SIGPLAN SIGOA symposium on Text manipulation*, pages 68–73, New York, NY, USA, 1981. ACM.

[9] Hyunmo Kang, Catherine Plaisant, and Ben Shneiderman. New approaches to help users get started with visual interfaces: Multi-layered interfaces and integrated initial guidance. In *National Conference on Digital Government Research*, volume 130, pages 1–6, 2003.

[10] MediaWiki. http://www.mediawiki.org/wiki/MediaWiki.

[11] Ben Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley, 3rd edition, 1998.

[12] Ben Shneiderman. Promoting universal usability with multi-layer interface design. In *ACM Conference on Universal Usability*, pages 1–8, 2003.

[13] Wikipedia. http://en.wikipedia.org/wiki/Main_Page.

[14] Wikipedia. *Potato*. http://en.wikipedia.org/wiki/Potato.

[15] Wikipedia. *Syntax highlighting*. http://en.wikipedia.org/wiki/Syntax_highlighting.

A Glossary

GUI Graphical User Interface — A user interface based on graphics and icons, rather than exclusively text

HCI Human-Computer Interaction — The field of study concerned with the interactions of humans and computers.

markup Additional text included in a body of natural text that provides extra information about the body of text.

Multi-Layered Design A design strategy in human-computer interaction where interfaces are divided into sequential layers of related functionality.

rich text Refers to text with additional formatting, such as boldface or italics.

syntax highlighting Automatic formatting of certain portions of text to emphasize syntactically significant features.

WYSIWYG What You See Is What You Get — A mode of editing rich text, where all formatting is shown in the editing area in its final form, rather then as markup. This behaviour is commonly found in word processors, such as Microsoft Word.

B User Survey

Participant Number: _____

I. Computer Experience

Please circle the letter corresponding to your answer to each question.

1. Have you used a computer before? If so, for how long?
   a. I have not used a computer
   b. Less than 1 year
   c. 1 to 2 years
   d. 2 to 4 years
   e. 4 to 6 years
   f. 6 to 8 years
   g. More than 8 years
   h. I decline to answer

2. In your opinion, which of the following best describes your computer skills?
   a. Beginner
   b. Intermediate
   c. Advanced
   d. I decline to answer

3. Have you used a word processor before? If so, for how long?
   a. I have not used a word processor
   b. Less than 1 year
   c. 1 to 2 years
   d. 2 to 4 years
   e. 4 to 6 years
   f. 6 to 8 years
   g. More than 8 years
   h. I decline to answer

4. How often do you use keyboard shortcuts when operating a computer?
   a. Never
   b. Rarely
   c. Occasionally
   d. Often
   e. Always
   f. I decline to answer

5. For each of the following technologies, have you used it, and if so, how often?
   5.1 HTML
   a. Never b. Occasionally c. Frequently d. I decline to answer

   5.2 Editors for Wikipedia or other wikis
   a. Never b. Occasionally c. Frequently d. I decline to answer

   5.3 Other markup languages (for instance, XML or LaTeX)
   a. Never b. Occasionally c. Frequently d. I decline to answer

   5.4 Computer programming languages
   a. Never b. Occasionally c. Frequently d. I decline to answer

II. Impressions of Test Program

Please circle the number which best reflects your agreement with the following statements, according to the following scheme:

1 = Strongly Disagree **2** = Disagree **3** = Neutral **4** = Agree **5** = Strongly Agree
NA = No Answer

6.	I found the program easy to use.	**1 2 3 4 5 NA**
7.	The program allowed me to complete the task quickly.	**1 2 3 4 5 NA**
8.	I would use this program in my normal work.	**1 2 3 4 5 NA**
9.	The buttons in the program clearly suggested their function.	**1 2 3 4 5 NA**
10.	The program clearly displayed the keyboard commands corresponding to each action.	**1 2 3 4 5 NA**
11.	The keyboard commands for each action were easy to remember.	**1 2 3 4 5 NA**
12.	The keyboard commands integrated naturally into the program's display.	**1 2 3 4 5 NA**

III. Freeform Answer

Please write your answer to each question in the space provided. You may decline to answer any question.

13. What, if anything, did you like about the test program?

14. What, if anything, could be improved in the test program?

15. Do you have any further comments about the program or study?

The questionnaire is completed. Thank you for your participation.