

Graphically Enhanced Keyboard Accelerators for GUIs

by

Jeff Hendy

B.Sc., The University of Arizona, 2007

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF  
THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA  
(Vancouver)

December 2009

© Jeff Hendy, 2009

## **Abstract**

We present the design and evaluation of Graphically Enhanced Keyboard Accelerators (GEKA), a user interface interaction method allowing commands within a graphical application to be quickly and easily invoked through the keyboard. The high-level goal of this work is to make interactive desktop computing more pleasant and productive for experienced computer users. GEKA is designed to provide complete coverage of the command set, to require low visual demand, and to support ease of learning and remembering, a low error rate, and high speed. This thesis describes GEKA's design and two related user studies.

A formative study with 10 participants explored how our target users currently work with Window, Icon, Menu and Pointer (WIMP) interfaces. The results of the study suggest that advanced computer users prefer to execute commands with the keyboard. However, they are often unable to do so in current applications because shortcuts are not available for all commands or are unknown. This indicates a desire among advanced users for a GEKA-like interaction method and motivates our research.

GEKA's design blends elements from WIMP and command line interfaces, allowing commands to be entered quickly and precisely while shifting the focus of the interaction to recognition rather than recall. GEKA has three key improvements over existing text command systems with graphical feedback: support for multiple parameters in arbitrary order, smarter matching – including abbreviations for all commands, and clear visual feedback of the input characters to facilitate learning and re-use.

A laboratory experiment with 12 participants compared GEKA to WIMP interaction methods. We found error rates to be nearly identical and speed to be very competitive. The experiment also explored users' preferences: When given a choice in situ between WIMP and GEKA for actual command execution, participants overwhelmingly used existing

keyboard shortcuts when they knew them and used GEKA when they didn't. In a questionnaire, each type of GEKA command was rated better than its WIMP equivalent except for zero-parameter GEKA commands relative to keyboard shortcuts. These results suggest that our target user population has a strong preference for GEKA interaction over the mouse-based WIMP methods.

# Contents

ABSTRACT .....	ii
CONTENTS .....	iv
LIST OF TABLES .....	vii
LIST OF FIGURES .....	viii
ACKNOWLEDGEMENTS .....	xi
DEDICATION .....	xii
<b>1 INTRODUCTION .....</b>	<b>1</b>
1.1 WINDOWS, ICONS, MENUS, AND POINTERS (WIMP) .....	2
1.1.1 <i>Menu bars</i> .....	2
1.1.2 <i>Toolbar buttons and drop-downs</i> .....	4
1.1.3 <i>Context menus</i> .....	5
1.1.4 <i>Keyboard shortcuts and mnemonics</i> .....	6
1.1.5 <i>Dialog boxes</i> .....	7
1.2 THE GUI GAP .....	8
1.3 GRAPHICALLY ENHANCED KEYBOARD ACCELERATORS .....	9
1.4 OVERVIEW OF THE THESIS .....	11
<b>2 RELATED WORK .....</b>	<b>12</b>
2.1 EXISTING INTERFACE PARADIGMS .....	12
2.1.1 <i>Graphical user interfaces</i> .....	12
2.1.2 <i>Command line interfaces</i> .....	14
2.2 COMPARING WIMP AND CLIS .....	16
2.3 OTHER APPROACHES TO WIMP AND CLI PROBLEMS .....	18
2.3.1 <i>Adaptable and adaptive GUIs</i> .....	18
2.3.2 <i>Search</i> .....	19
2.3.3 <i>Improvements to command line interfaces</i> .....	21
2.4 LITERATURE SUPPORTING KEYBOARD INTERACTION .....	22
2.5 RELATED APPLICATIONS .....	22

<b>3</b>	<b>FORMATIVE STUDY .....</b>	<b>26</b>
3.1	PHASE 1 TASKS .....	27
3.2	PHASE 2 TASKS .....	30
3.3	APPARATUS .....	32
3.4	PARTICIPANTS .....	32
3.5	PROCEDURE .....	33
3.6	RESULTS .....	33
3.6.1	<i>Phase 1</i> .....	33
3.6.2	<i>Phase 2</i> .....	38
3.7	LIMITATIONS .....	40
3.8	DISCUSSION AND CONCLUSIONS.....	41
<b>4</b>	<b>GEKA DESIGN AND PROTOTYPES .....</b>	<b>42</b>
4.1	GOALS.....	42
4.1.1	<i>GEKA vision</i> .....	42
4.1.2	<i>Prototype goals</i> .....	43
4.1.3	<i>Long term goals</i> .....	44
4.2	PROTOTYPE DESIGN .....	45
4.2.1	<i>GEKA command language</i> .....	46
4.2.2	<i>GEKA graphical feedback</i> .....	49
4.3	DESIGN DISCUSSION .....	62
4.4	DESIGN LIMITATIONS.....	63
<b>5</b>	<b>LABORATORY EXPERIMENT .....</b>	<b>68</b>
5.1	EXPERIMENTAL ENVIRONMENT AND TASKS.....	68
5.1.1	<i>Commands</i> .....	69
5.2	TASK .....	71
5.3	PARTICIPANTS .....	74
5.4	PROCEDURE .....	75
5.5	PILOTING .....	76
5.6	DEPENDENT MEASURES .....	77
5.7	MOTIVATION .....	78

5.8	HYPOTHESES.....	79
5.9	DESIGN .....	79
5.10	RESULTS .....	80
5.10.1	<i>Time</i> .....	80
5.10.2	<i>Errors</i> .....	85
5.10.3	<i>Method choice</i> .....	85
5.10.4	<i>Qualitative findings</i> .....	86
5.10.5	<i>Summary of results</i> .....	90
5.11	DISCUSSION .....	90
<b>6</b>	<b>CONCLUSIONS AND FUTURE WORK .....</b>	<b>93</b>
6.1	DISCUSSION AND LIMITATIONS .....	95
6.2	FUTURE WORK .....	98
	<b>BIBLIOGRAPHY .....</b>	<b>101</b>
	<b>APPENDIX A FORMATIVE STUDY DOCUMENTS.....</b>	<b>105</b>
	<b>APPENDIX B LAB EXPERIMENT COMMAND IMAGES... </b>	<b>107</b>
	<b>APPENDIX C EXPERIMENT QUESTIONNAIRE .....</b>	<b>111</b>
	<b>APPENDIX D ETHICS APPROVAL CERTIFICATES .....</b>	<b>123</b>

## List of Tables

Table 2-1: An overview of command language structures. ....	15
Table 3-1: Commands used in the interview portion of our formative study. ....	28
Table 3-2: Commands used in dialog box exploration during our formative study. ....	30
Table 3-3: Commands used in GEKA practice during Phase 2 of our formative study. ....	30
Table 3-4: Commands used during the guided task in Phase 2 of our formative study. ....	31
Table 3-5: Command usage in Word 2003. ....	34
Table 3-6: Keyboard versus mouse usage for the six command executions involving parameters. ....	37
Table 4-1: Examples of GEKA commands. ....	47
Table 5-1: The WIMP methods and their GEKA equivalents. ....	69
Table 5-2: Breakdown for the interaction between condition, block, and interaction method. ....	84
Table 5-3: Total errors. ....	85

## List of Figures

Figure 1-1: A menu bar in Mac OS X.....	3
Figure 1-2: Two toolbars in a Mac OS X application.....	4
Figure 1-3: A context menu in a Mac OS X application .....	5
Figure 1-4: A dialog box in a Windows application.....	7
Figure 2-1: Mac OS X's Spotlight search mechanism.....	20
Figure 2-2: Mozilla Firefox 3 showing the search mechanism built into the address bar. ....	20
Figure 2-3: The search mechanism built into the Help menu of Mac OS X 10.5.....	21
Figure 2-4: Quicksilver .....	23
Figure 2-5: Enso.....	24
Figure 3-1: Preferences for each WIMP method .....	35
Figure 3-2: Percentage of commands reported to be most frequently executed, with each technique broken down by frequency of command use .....	36
Figure 3-3: Likert scale responses from Phase 2 of the formative study .....	39
Figure 4-1: GEKA prototype graphical feedback .....	51
Figure 4-2: GEKA prototype after the characters "p c " have been entered...54	54
Figure 4-3: The result of pressing BACKSPACE in Figure 4-2.....	54
Figure 4-4: Initial GEKA command list with all possible commands.....	55
Figure 4-5: GEKA command list after the character 's' has been typed. ....	56
Figure 4-6: GEKA command list after the characters 'tab' have been typed. The best matching command is insert table. This command list contains all four categories of match described in the command language section. ....	56
Figure 4-7: GEKA command list after the characters 'align' have been entered. There is no way to select a command other than center alignment by continuing to type characters. ....	57
Figure 4-8: GEKA command list after the characters 'align' have been entered and the down arrow has been pressed twice. ....	58
Figure 4-9: GEKA window with the parameter list in focus .....	59
Figure 4-10: GEKA window showing the parameter list for 'insert table' after 'c' has been entered.....	59
Figure 4-11: GEKA window with the parameter value entry in focus .....	60



Figure 4-12: GEKA window with the parameter value entry in focus .....	61
Figure 4-13: GEKA window demonstrating the text entry area .....	62
Figure 4-14: The Find and Replace dialog in Microsoft Word 2008.....	64
Figure 4-15: The font colour chooser in Microsoft Word 2008 .....	65
Figure 4-16: The “format paintbrush” item in Microsoft Word 2008 .....	65
Figure 4-17: The print dialog box in Microsoft Word 2003 .....	66
Figure 5-1: Command image for 'bold' .....	71
Figure 5-2: Command image for 'copy' .....	72
Figure 5-3: Command image for 'font size 24'.....	72
Figure 5-4: Command image for 'print copies 2 pages selection' .....	72
Figure 5-5: Experimental application .....	74
Figure 5-6: Dialog box times for WIMP and GEKA in each block .....	81
Figure 5-7: Menu bar times for WIMP and GEKA in each block.....	82
Figure 5-8: Keyboard shortcut times for WIMP and GEKA in each block....	82
Figure 5-9: Toolbar button times for WIMP and GEKA in each block .....	83
Figure 5-10: Toolbar drop-down times for WIMP and GEKA in each block .....	83
Figure 5-11: Percentage of command executions using each method in the method choice phase of the experiment.....	86
Figure 5-12: Ratings for WIMP and GEKA methods from the qualitative feedback phase of the experiment, part 1 of 2 .....	88
Figure 5-13: Ratings for WIMP and GEKA methods from the qualitative feedback phase of the experiment, part 2 of 2 .....	89
Figure A-1: Unformatted document provided to formative study participants. .....	105
Figure A-2: Formatted document for formative study participants to match. .....	106
Figure B-1: Lab experiment image for “bold” .....	107
Figure B-2: Lab experiment image for “italic” .....	107
Figure B-3: Lab experiment image for “paste” .....	107
Figure B-4: Lab experiment image for “font size 24” .....	108
Figure B-5: Lab experiment image for “underline” .....	108
Figure B-6: Lab experiment image for “save” .....	108

Figure B-7: Lab experiment image for “print copies 3 page range selection” .....	108
Figure B-8: Lab experiment image for “insert table rows 5 columns 3” .....	109
Figure B-9: Lab experiment image for “insert page numbers position top alignment center first page no” .....	109
Figure B-10: Lab experiment image for “undo” .....	109
Figure B-11: Lab experiment image for “apply style heading 1” .....	109
Figure B-12: Lab experiment image for “center” .....	110
Figure B-13: Lab experiment image for “toggle bullets” .....	110
Figure B-14: Lab experiment image for “line spacing 2” .....	110
Figure B-15: Lab experiment image for “copy” .....	110

## **Acknowledgements**

The supervisory committee included co-supervisors Dr. Kellogg S. Booth and Dr. Joanna McGrenere, and Dr. Michiel van de Panne who served as a “second reader” for the thesis.

The research reported in this thesis was supported by funding from the Natural Sciences and Engineering Research Council of Canada under the Strategic Research Network program through the Network for Effective Collaboration Technology through Advanced Research, and under the Discovery Grant program. Research facilities were provided under funding from the Canada Foundation for Innovation and the British Columbia Knowledge Development Fund for the Institute for Computing, Information & Cognitive Systems at the University of British Columbia.

## **Dedication**

To East Van Bike Polo for giving me a family and a home in this foreign land. Mistey eyed X100. 1-2-3 KILL!

# 1 Introduction

This thesis documents research on the design and evaluation of Graphically Enhanced Keyboard Accelerators (GEKA), an interaction method designed to allow fast and easy-to-learn interaction with interactive desktop computer applications. Most computers today make use of a graphical user interface (GUI), which is any interface that has visual feedback beyond plain text. GEKA addresses the problems of a specific type of GUI called Windows, Icons, Menus, and Pointers (WIMP). WIMP interfaces organize applications into windows and allow commands to be executed through menus and buttons. This is the main type of interface on nearly all major operating systems.

GEKA is designed to address a specific problem with WIMP interaction, namely that its two distinct types of interaction leave experienced computer users with an unmet need. The first type of WIMP interaction is mouse-based, which is easy to learn but slow to use. The second type is keyboard-based interaction, which is fast to use but difficult to learn and incomplete. We call the unexplored design space between these two extremes “the GUI gap.” GEKA is designed to fill the GUI gap by providing a keyboard-based interaction method that is both quick to use and easy to learn while being available for nearly all commands within an application. GEKA accomplishes this by combining features from modern graphical interfaces and traditional command line interfaces (CLIs).

In this chapter, we explore WIMP interaction, describing the interaction methods that comprise WIMP and discussing their shortcomings. We then introduce the concept of the GUI gap, describe how our work draws from CLIs, and then discuss GEKA at a high level. The chapter concludes with an overview of the research and the rest of this thesis.

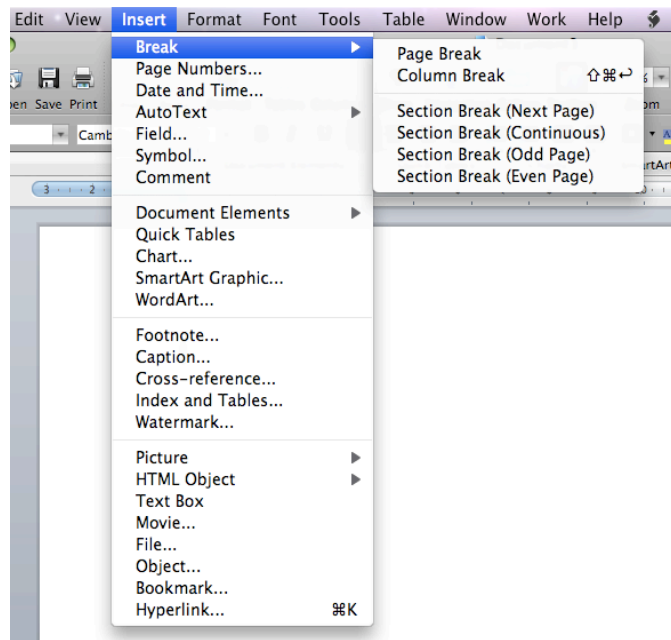
## **1.1 Windows, Icons, Menus, and Pointers (WIMP)**

Nearly all of today's personal computers use a graphical user interface based on the WIMP paradigm. Applications are contained in windows, which can be manipulated with a pointing device that is typically a mouse. There are several methods for executing commands within a WIMP application, including menu bars, toolbar buttons, toolbar drop-downs, context menus, keyboard shortcuts, and mnemonics. We refer to these, as well as dialog boxes, which are often used to select parameters for commands, as the WIMP methods.

This section briefly describes each of the WIMP methods in order to allow for discussion of the GUI gap and GEKA in subsequent sections. Most readers will already be familiar with these methods and may only require a brief skim of this section. Detailed information about these methods and how to properly use them appears in Shneiderman's *Designing the User Interface* (1997). Chapter 2 in this thesis discusses literature on WIMP interfaces and specific design elements that have influenced GEKA.

### **1.1.1 Menu bars**

Menu bars, which are typically located either at the top of the screen (Macintosh) or at the top of each window (Windows and Linux), organize commands into a series of drop-down menus. The menus typically contain all of the commands available in an application. The commands are organized into high-level categories on the menu bar, and often further organized into sub-menus. Figure 1-1 shows a menu bar from an application in Mac OS X 10.5.



**Figure 1-1: A menu bar in Mac OS X. The menu bar is at the top of the screen, and each of the top level menus can be pulled down from the bar with a mouse click. Some menus contain submenus, such as the “Break” submenu located within the “Insert” menu shown in this example.**

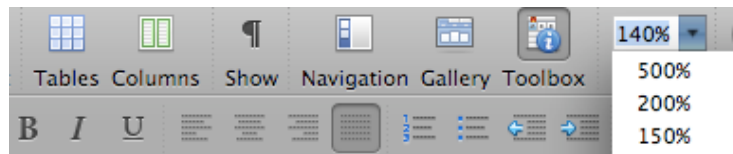
Menu bars are a space-efficient way to organize all of an application’s commands. They take very little space when the menus are all closed but can expand to accommodate huge numbers of commands. Menu bars can be useful for learning about an application. A new user can quickly scan through the commands listed in each menu to get an idea of what types of actions are possible. The hierarchical structure of menu bars provides an organization of commands that can make them easy to find. The plain text format of menu items makes it possible to concisely describe commands so that there is little ambiguity about an item’s function.

On the downside, menu bars can be quite slow and frustrating to use. In the best case, when a user knows exactly where the desired menu item is, there are still two mouse clicks required: one to open the menu, and one to select the item. This is often frustrating because mouse clicks require precise motor movement and full visual attention. Menu bars are considered to be quite slow by experienced computer users. Furthermore, the organization of

the menu hierarchy may not match the user's mental model, making it difficult to find the desired item. In today's complex applications, this can result in the need to scan through dozens of menu items before the desired one is found.

### 1.1.2 Toolbar buttons and drop-downs

Toolbars are typically located at the top of each window and contain two distinct interaction methods: buttons, which are selected with a simple click, and drop-downs, in which a value is selected from a list of alternatives. Figure 1-2 shows a portion of the toolbar from a typical Mac OS X 10.5 application, which includes both buttons and drop-downs.



**Figure 1-2: Two toolbars in a Mac OS X application. The top toolbar includes buttons with both icons and text, while the bottom toolbar has icons only. Additionally, there is a drop-down menu on the right side of the top toolbar, which is currently opened.**

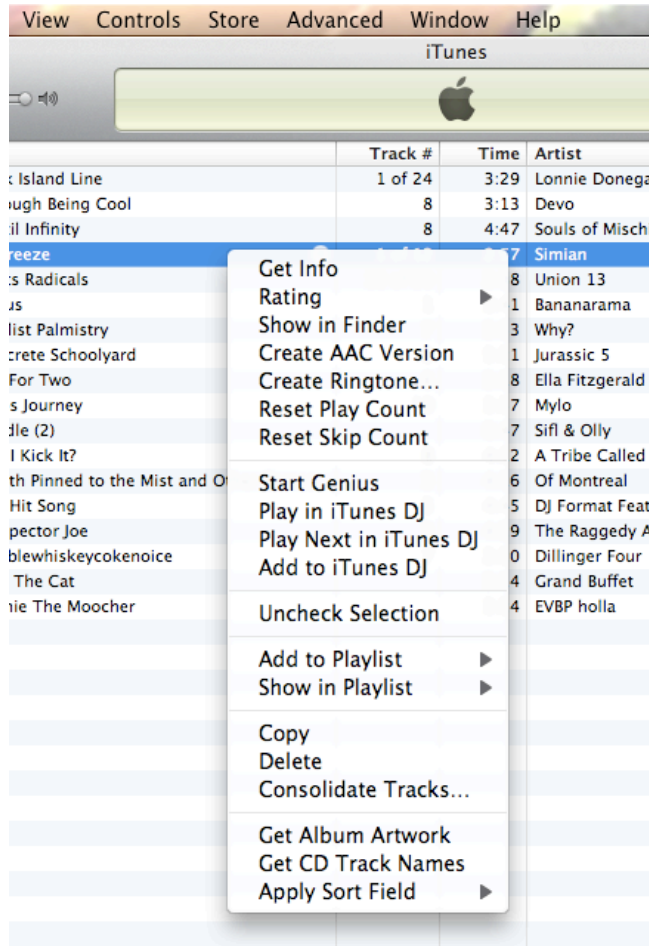
Toolbars provide easy access to the most frequently used commands in an application. Toolbar buttons are always visible and can thus be selected with only one click. Because toolbars are always visible, they can also be used to provide state information. For example, two of the buttons in Figure 1-2 are drawn as being depressed, showing which setting is currently selected. The drop-down also shows its current value.

Toolbar items, while generally being easier to access than menu bar items, still require the use of the mouse, which frustrates many users. Because toolbars are always visible, they typically contain only a small number of commands in order to preserve screen space. Visual icons on the toolbar can often be difficult to interpret, sometimes requiring the user to hover the mouse above the icon to read the textual “tooltip” describing the icon’s functionality. This can be mitigated by adding text to the icon, as seen in the top toolbar of Figure 1-2, but doing so takes up even more screen space.



### 1.1.3 Context menus

Context menus pop up when the secondary mouse button is clicked (typically a right click) within an application. Figure 1-3 shows a context menu from a typical Mac OS X application.



**Figure 1-3: A context menu in a Mac OS X application. This menu appears when the right mouse button is clicked on an item within the application and lists commands that can be performed on that item.**

Context menus generally contain only commands that can be executed on the currently selected item. For this reason, they are often a good way to find the desired command because there are far fewer items to scan than there are with the menu bar. However, context menu item selection still requires at least two mouse clicks.

#### 1.1.4 Keyboard shortcuts and mnemonics

The previously described methods are all mouse-based. Experienced users often desire to execute commands quickly without the need to use the mouse. Currently there are two choices for doing so in WIMP interfaces: shortcuts (also known as hotkeys) and mnemonics.

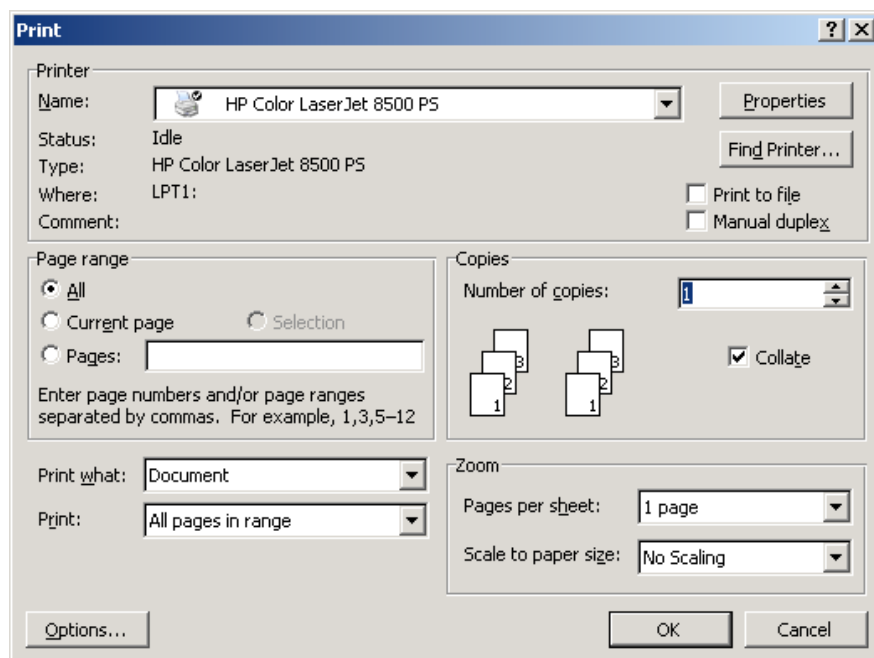
Most WIMP platforms support some form of keyboard shortcut to execute commands. A modifier key (generally **CTRL** on Windows and **COMMAND** on Mac) is pressed in combination with one or more other keys to execute a command. This method can be very useful for experienced users, because pressing a two-key combination is generally much faster than a mouse selection and it requires no visual attention.

When the number of commands is small, simple and intuitive mappings can be created for keyboard shortcuts such as **CTRL+B** for **bold** or **CTRL+S** for **save** in Microsoft Word. However, when there are more than just a handful of commands, straightforward mappings are not always possible, resulting in combinations such as **CTRL+SHIFT+P** for **superscript** in OpenOffice.org and **CTRL+ALT+SHIFT+L** for **auto contrast** in Adobe Photoshop CS4. Because of this, keyboard shortcuts are generally not available for all commands in an application and, beyond a small basic subset, are often quite difficult to learn and remember even when they are available.

Microsoft Windows has another form of keyboard support: mnemonics. Each item in a menu or dialog box has a single underlined character. The item can be accessed by pressing **ALT** and then the underlined character. For example, in many programs in Windows, the **save** command can be executed by pressing **ALT** to enter mnemonic mode, followed by **F** to open the file menu and then **S** to select the **save** item. As with keyboard shortcuts, mnemonics can often save time, but the mappings can become quite unnatural when there are a large number of commands. For example, the combination to access **cut** is typically **ALT -> E -> T**.

### 1.1.5 Dialog boxes

When a command in a WIMP application requires parameters, the parameters are usually chosen through a dialog box. Figure 1-4 shows a typical dialog box from an application in Microsoft Windows. The desired values can be chosen through a set of widgets that might include text boxes, check boxes, radio buttons, sliders, and more. When the values have been chosen, the command can generally be executed by clicking a button at the bottom of the dialog box, in this case the **OK** button.



**Figure 1-4:** A dialog box in a Windows application. This dialog box allows the user to enter parameters for the “print” command. Pressing “OK” executes the command with the selected parameters.

Dialog boxes generally do a good job of displaying all possible options in a logical format that makes it easy to find and select the desired parameters. As with the other methods described above, dialog boxes have problems when there are a large number of options. Often the only way to fit all of the options into a single dialog box is to use multiple tabs. This adds more steps to the selection process and can make it difficult to find the desired parameter if the tabs are not organized the way that the user would expect them to be.

Dialog boxes are typically intended to be used with a pointing device, but can be navigated with the keyboard as well. Typically, the **TAB** key moves between the various parameters in a dialog box, and the individual values can be specified by typing (for text boxes) or with the arrow keys (for menus and radio buttons). This is very helpful for small simple dialog boxes. For example, in Microsoft Word, to insert a table with 4 rows and 2 columns, once the **insert table** dialog box is open, the user could simply type **4 TAB 2 ENTER** and be done. However, this interaction can be problematic when there are a large number of options. Sometimes the **TAB** key needs to be pressed so many times to reach the desired element that any benefits of using the keyboard are negated. Worse, the **TAB** key navigation is sometimes not linear (top-down, left-to-right) in large dialog boxes, and it is often impossible to know where the focus will move on the next **TAB** press.

## 1.2 The GUI gap

The WIMP methods described above fall into two distinct categories: the mouse-based methods of menu bars, toolbars, and context menus, and the keyboard-based methods of shortcuts and mnemonics. The mouse-based methods are what make WIMP so easy to learn and use that it has become the primary interaction technique on most personal computers. Before GUIs and WIMP interfaces, most computing was done through CLIs, which require the user to learn complicated syntax and memorize many obscure command names. This essentially restricted computer use to highly trained experts. The visual nature of the mouse-based WIMP methods make the basis of interaction recognition, rather than recall, which makes these methods easy to learn, use, and remember even for people with little to no computer experience or training (Norman, 1988). Even for advanced computer users who are proficient with CLIs, these mouse-based methods are very helpful when learning a new application or accessing infrequently used commands.

Mouse-based techniques, of course, are not without problems. Advanced users often know exactly what command they want to execute and exactly where it is, but are slowed down by the need to position the pointer. If the users' hands are not in the proper position for pointing, they must first be repositioned. Once the hands are in place, pointer positioning, whether it be mouse-, finger-, or stylus-based requires physical movement that is governed by Fitts's Law (MacKenzie, 1995). Especially for the mouse, pointer movement requires the user's visual attention to be diverted from the main task to follow the pointer, which can be quite distracting.

These issues are partially addressed by the keyboard-based WIMP methods, which allow experienced users to quickly execute commands with little distraction. However, these methods are not available for all commands and when they are available, they often remain unknown to users because of difficulty in learning and remembering them. Furthermore, when a command has parameters, the keyboard shortcut or mnemonic brings up a dialog box. It is possible to navigate a dialog box with the keyboard, but, as discussed above, this is often confusing and tedious.

In short, the mouse-based methods are easy to learn and remember, but are slow and frustrating for advanced users, while the keyboard-based methods are fast but not available for all commands and often too difficult to learn even when they are available. We call the unexplored design space between these two extremes the GUI gap. We are working toward filling this gap with a new interaction method that is easy to learn and remember, available for almost all commands and parameters, and faster than mouse-based methods.

### **1.3 Graphically Enhanced Keyboard Accelerators**

We have designed an interaction method called Graphically Enhanced Keyboard Accelerators (GEKA) that aims to fill the GUI gap. GEKA's

primary goal is to make computing more pleasant for advanced computer users. We expect to achieve this through a combination of high speed, low error rate, and easy-to-learn-and-remember commands. Even if we cannot demonstrate an advantage in any of these quantifiable areas, we expect that many people will choose to use GEKA simply because they prefer the keyboard and are currently not able to use it for many tasks. We focus on advanced computer users because we consider them the most likely to adopt and benefit from a new keyboard-based interaction method. However, our design is not intended to disadvantage less experienced users. We expect them to share many of the same benefits that advanced users get from GEKA.

GEKA's design is influenced heavily by traditional CLIs. In a CLI, command names are typically abbreviated to a few characters, allowing them to be typed very quickly by an experienced user, and parameters can be very powerful and flexible, with the user able to set values for only the relevant parameters and to do so in any order. Additionally, many CLIs have auto-completion mechanisms for commands and parameters, which reduce the amount of typing necessary. For an experienced user, a CLI provides an environment where desired actions can be performed very quickly with little distraction from the main task. The biggest downside of CLIs is that the user must memorize all necessary command and parameter names (or tediously look them up each time they are used). This makes it very difficult to learn and very easy to forget how to use commands in a CLI.

GEKA uses the flexible syntax and time saving auto-completion of a CLI while drastically reducing the amount of memorization required by using natural, plain English command names and making use of graphical feedback to display command and parameter options, thus shifting the focus of the interaction to recognition rather than recall.

Incorporating these concepts into a GUI is not an entirely new idea. Auto-completion is seen in places such as the formula editor in Microsoft Excel, the address bar of a web browser, and function names in IDEs. A

handful of programs, including Enso (Humanized, n.d.) and Quicksilver (Blacktree, n.d.), allow a limited number of commands to be executed through the keyboard in a GUI environment, incorporating auto-completion and graphical feedback. These programs are discussed in more detail in Chapter 2.

## **1.4 Overview of the Thesis**

This thesis documents the design and evaluation of our first version of GEKA. Chapter 2 discusses previous work related to this research, including evaluations of and comparisons between WIMP and CLIs, other approaches to improving user experience with WIMP and CLIs, and previous applications that combine graphics with keyboard commands. Chapter 3 discusses a formative study, which examined how advanced computer users interact with current WIMP interfaces and which found support for our notion of the GUI gap. The formative study included a preliminary evaluation of an early GEKA prototype. Chapter 4 presents the design of our current GEKA prototype, discussing how it functions and our design rationale. Chapter 4 also explains the changes to our design that were made as a result of the formative study. Chapter 5 documents a laboratory experiment evaluating the GEKA prototype. The experiment compared advanced users' performance in GEKA and WIMP with respect to time and error rate. It also explored their preferences between GEKA and WIMP interaction methods. Chapter 6 discusses directions that future research on GEKA could take and summarizes conclusions from the research.

## 2 Related Work

This chapter is an overview of the literature and applications that are relevant to the research. We begin by discussing current WIMP and command line interfaces (CLIs), describing how each of them has influenced GEKA’s design and then summarizing several papers that compare WIMP with CLIs. Next, we cover alternate approaches to mitigating the problems that are inherent in WIMP and CLIs. We then discuss literature by prominent members of the HCI community pointing out that a GEKA-like interface is needed. We conclude by describing existing applications that have inspired the design of GEKA.

### 2.1 Existing interface paradigms

GEKA runs within WIMP-based GUI environments. Many aspects of GEKA’s design are inspired by features of CLI environments. As such, it is important to discuss these two interface paradigms. We describe the relevant features of each. Further thoughts on how to properly design command line and graphical interfaces can be found in Shneiderman’s *Designing the User Interface* (1997).

#### 2.1.1 Graphical user interfaces

Shneiderman (1983) provides an overview of early GUIs and describes their benefits. Among them are “novices can learn basic functionality quickly,” “intermittent users can retain operations concepts,” “error messages are rarely needed,” and “users can immediately see if their actions are furthering their goals.” All of these benefits derive from the fact that the use of graphics allows the application to show a detailed visual depiction of the



work being performed as well as a listing of all the possible actions that the user can choose from.

Since then, nearly all personal computer systems have adopted a WIMP paradigm and users have come to expect the benefits described above. Shneiderman (1997) later enumerated eight “golden rules” for designing user interfaces. Among them are “offer informative feedback,” “design dialog to yield closure,” “offer simple error handling,” “permit easy reversal of actions,” and “reduce short-term memory load.” The latest version of Apple’s “Human Interface Design Principles” (Apple Computer, 2009) includes many similar ideas such as “direct manipulation,” “feedback and communication,” “what you see is what you get,” and “forgiveness.” Most other sets of user interface guidelines include some variant of these rules. Each of these are either much easier with or only possible because of the use of graphics.

GEKA exists in a graphical environment. It is thus capable of capturing the benefits and adhering to the guidelines described above. Indeed, it must do so in order to meet the expectations of today’s users.

Most GUIs provide keyboard shortcuts that allow efficient interaction for experienced users. Recent work, however, has shown that keyboard shortcuts are underutilized even by experienced computer users. Lane, Napier, Peres, & Sándor (2005) conducted a study of 251 professional workers who used Microsoft Word frequently as part of their jobs. For many of the most frequently used commands in Word, participants were asked what percentage of the time they used each possible method of execution, including shortcuts, toolbar, menu items, and mnemonics. There were no commands found in which keyboard shortcuts were the most used method.

Peres, Tamborello, Fleetwood, Chung, & Paige-Smith (2004) explored social factors that might lead to low shortcut usage with 82 participants, finding that shortcut usage was much more common among participants who watched others use computers and who knew other people who used shortcuts. The participants who reported low shortcut usage provided

questionnaire responses indicating that they would be reluctant to start using shortcuts even if they felt that it would save them time and somebody was available to train them on the shortcuts. Grossman, Dragicevic, & Balakrishnan (1997) explored ways to encourage shortcut usage such as displaying the shortcuts more prominently in menus and disabling frequently used menu items to force shortcut usage.

We were surprised by Lane, Napier, Peres, & Sándor's (2005) findings of such low shortcut usage because it does not match our personal experience working in an environment populated largely by experienced computer users. We conducted a formative study, described in Chapter 3, to examine, among other things, shortcut usage among technical graduate students in order to help explore this issue.

### **2.1.2 Command line interfaces**

There are two key syntactic properties in describing a command language. The first is the order of tokens: in a prefix language, the command comes before the parameters, whereas in a postfix language, the command comes after the parameters. The second property is parameter specification: with positional parameters, all parameters must be specified in order. With keyword parameters, only the needed parameters have to be specified, and the user can do so in any order.

Buxton (1982) discussed the issue of prefix vs. postfix in a paper on selection-positioning tasks. Cherry (1986) conducted a study with 60 participants that found no performance difference between a prefix and a postfix language, though participants indicated a preference for the prefix language. There do not appear to be any studies comparing positional to keyword parameters.

Table 2-1 provides an overview of the command language structures with examples of languages from the categories where we know one to exist. GEKA uses prefix commands with keyword parameters.

**Table 2-1: An overview of command language structures. GEKA is a prefix, keyword language.**

	Positional parameters	Keyword parameters	Mixed parameters
Prefix	Ubiquity	GEKA	UNIX, JCL
Postfix	HP-10 calculator		
Mixed/infix	Quicksilver		

Our work builds on a number of features that existed in pre-GUI CLI implementations. We mention only a few highlights. OS/360 introduced JCL, perhaps the most complex CLI to date, with a myriad of commands, parameters, and optional specifications. Like JCL, the OS/360 macro assembler language accepted both positional and keyword parameters. Keyword parameters allowed for shorter specifications because parameters whose default values were appropriate do not need to be listed.

The original command completion feature on the UC Berkeley-developed CLI for the SDS 940 Genie operating system was automatic – as soon as the stem uniquely determined the command the full command name was typed by the system. This was later modified for the PDP-10 Tenex CLI so that command completion only took place when **ESC** was typed, and this was extended as well to provide file name completion. This led to **TAB** completion in **tcsh** on Unix, which also provides a list of possible completions to provide recognition-based hints to the user if **CTRL-D** is typed instead of **TAB** (Wikipedia, 2009).

There is a clear pattern in the development of traditional CLIs. As the complexity of the CLI increased, features were introduced to decrease the number of keystrokes required to specify a command and its parameters. In some cases (such as **tcsh**) visual aids were added (the list of possible completions) to allow users to rely on recognition rather than recall.

Command naming has always been an issue with CLIs. Several papers have been published on the matter, with the most relevant being a study by Grudin and Barnard (1985). This study examined four groups of seven participants learning command names in one of four conditions: one in which all commands used abbreviated names chosen by the researchers, one in which participants could use full command names or the researchers' abbreviations, one in which participants started with the full command names and later moved to the researchers' abbreviations, and one in which participants started with the full command names and later moved to abbreviations that they chose themselves. The participants who began the study using full command names made fewer errors at the beginning of the experiment. After switching to abbreviations, those who were using the researchers' abbreviations continued to outperform those who were using abbreviations all along. Performance for those who created their own abbreviations actually got worse when they began using the abbreviations.

As with the more advanced CLIs, GEKA relies on auto-completion and visual aids to make interaction faster and easier. GEKA commands use the full names that are found in menus and dialog boxes, but we provide a built-in abbreviation for each command and an auto-completion mechanism that allows other character sequences to become abbreviations. Based on Grudin and Barnard's findings, this should help reduce error rates when using GEKA.

## **2.2 Comparing WIMP and CLIs**

Many studies compare elements of CLI and WIMP interaction. We discuss several of these studies here. These are not always direct comparisons between a full CLI and a pointer-based WIMP interface. For example, one study compared keyboard menus to command lines. While more direct comparison would be ideal, each of these studies does compare some aspect of CLI to some aspect of WIMP and is therefore useful in determining how to blend the two interaction techniques in GEKA.

Gong and Salvendy (1995) conducted an experiment in which 40 participants completed identical tasks with one of four interface types: keyboard menu-based, text command-based, hybrid (offering both menu and command options), and adaptive (initially the same as the hybrid, but forcing command usage as participants gained experience). At the beginning of the experiment, the menu-based interface was the fastest, but by the end, the adaptive and command-based interfaces dominated. In Likert-based user satisfaction ratings, the command-based interface fared worse than the menu-based on all questions, though these differences were not statistically significant.

Geller and Lesk (1983) compared user preference between menu and command/search based interfaces for two information retrieval tasks: finding information about books in a library catalog and reading the daily news. For each task users chose between a hierarchical menu containing all entries and a keyword-based search mechanism. Over 900 users accessed the library catalog during the study. These users preferred the keyword search because they typically knew what they were looking for and the search was much more efficient than navigating the large hierarchy in the menu. Between 100 and 150 people had access to the news program. These users preferred the menu because there was no way of knowing what to search for given that news is very dynamic.

Whiteside, Jones, Levy, & Wixon (1985) compared file manipulation tasks with 76 participants on three types of interfaces: command, iconic, and menu. Performance was found to be best on the command-based interfaces for all experience levels of the participants. Despite this, the most popular interface was iconic.

Westerman (1997) explored individual differences in using command and menu interfaces among 64 participants. There was no performance or preference difference between interfaces based on cognitive ability as measured by tests of verbal ability, spatial memory, spatial visualization,

logical reasoning, and associative memory. Additionally, there was no significant difference in preference based on experience level.

Karat, McDonald, & Anderson (1986) compared menu selections using mouse, keyboard, and touch panel. In all cases, the mouse was the least preferred of the three methods. The touch interface consistently had the highest performance. In one experiment with 24 participants, keyboard outperformed mouse. A second experiment, which was identical except that there were 48 participants and they were given more practice trials, showed the reverse outcome.

Most of these studies showed that a more CLI-like (keyboard or command) interface was faster than a more WIMP-like (mouse, menu, or icon) option. Preference was less clear and appears to depend largely on the task involved. These findings are encouraging, showing that GEKA has a good chance of outperforming WIMP interaction, but that it will need to be designed carefully and put into an appropriate context for users to react positively.

## **2.3 Other approaches to WIMP and CLI problems**

The problems that we are addressing in both WIMP and CLIs are well known. Several different approaches have been pursued to address them. We summarize some of those approaches.

### **2.3.1 Adaptable and adaptive GUIs**

One approach to the problem of icons and menu items often being difficult to locate in WIMP applications is to alter the interface so that the most likely to be needed items are in the easiest to use locations. There are two main approaches to doing this: adaptable interfaces in which the user decides when and where to move interface elements, and adaptive interfaces in which the application controls these decisions. Examples of adaptable interfaces are the Buttons system (MacLean, 1990) that allows the user to

place buttons with custom functionality within a UNIX desktop environment, and user interface Facades (Stuerzlinger, Chapuis, Phillips, & Roussel, 2006) which allow users to move buttons, sliders and other interface elements to any window, toolbar, or dialog box that they see fit. McGrenere, Baecker, & Booth (2002) evaluated an adaptable interface that allows users to simplify the interface in Microsoft Word by removing elements, finding that it was well utilized and well received. Examples of adaptive interfaces include the adaptive condition in the previously mentioned study by Gong and Salvendy (1995) that forces users to move from menu use to command use after a certain level of expertise, and SUPPLE (Gajos & Weld, 2004), an application that automatically adds frequently used features to a second pane within a user interface. A compromise between adaptable and adaptive interfaces is the mixed-initiative approach in which the application suggests changes to the interface, but the user must approve them. An example of this is the MICA system (Bunt, Conati, & McGrenere, 2007), which builds upon McGrenere et al.'s (2002) adaptable system.

While these systems can often make WIMP interfaces easier to manage and more efficient to use, they do not address the core problem that we are dealing with. The available interaction methods do not change, and thus the GUI gap remains.

### **2.3.2 Search**

As shown in the study by Geller and Lesk (1983), described above, users prefer search mechanisms over menus when they know what they are looking for and the set of choices is large. GUI designers have begun to acknowledge this fact and build convenient search mechanisms into software. Microsoft Windows and Mac OS X now both include file searches that can be opened with one or two keystrokes to very quickly search through an index of all files on the computer. Figure 2-1 shows the Mac OS X search mechanism. Most web browsers have a feature that displays relevant items from the user's browsing history when text is typed into the address bar. Mozilla Firefox 3

goes a step beyond this by including the user's bookmarks in this search. This is shown in Figure 2-2. Microsoft has released a plug-in for Office 2007 (Microsoft, 2009) that searches through all of the commands available in the ribbon, displaying those that match the input and allowing them to be executed by a simple mouse click or key press. In Mac OS X 10.5, there is a search box in the Help menu for each application that searches both the help contents and the contents of the application's menu bar, dynamically changing the contents of the help menu to reflect the search. Figure 2-3 shows this search box.

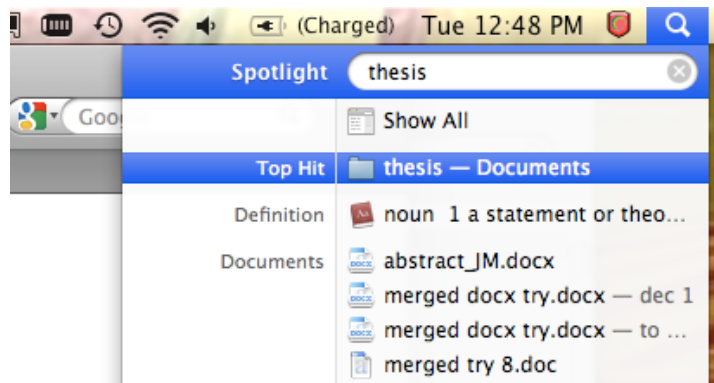


Figure 2-1: Mac OS X's Spotlight search mechanism.

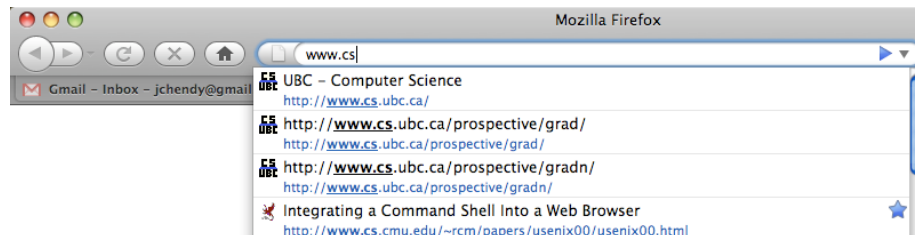
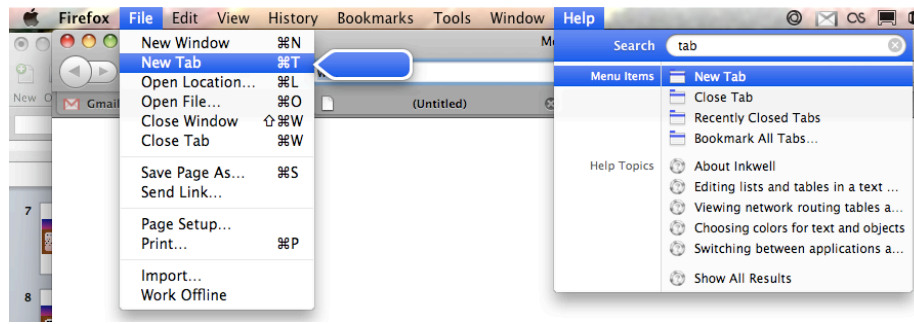


Figure 2-2: Mozilla Firefox 3 showing the search mechanism built into the address bar.





**Figure 2-3: The search mechanism built into the Help menu of Mac OS X 10.5.**

Each of these search mechanisms uses an incremental search, which updates the results as each character is typed as opposed to having to enter the full search term before any results are displayed. Raskin strongly advocated for incremental search in his book *The Humane Interface* (Raskin, 2000). GEKA's auto-completion mechanism uses incremental search.

### **2.3.3 Improvements to command line interfaces**

While CLIs have not dramatically changed in several decades, there have been some improvements. Examples in the literature include automatically generating optimal command aliases (Nichols & Ritter, 1995), mechanisms to more easily locate files by searching for file attributes (Giger & Wilde, 2006), and more efficient history mechanisms (Greenberg & Witten, 1988). There has also been some work done on creating an adaptive command line that predicts the next desired command (Davison & Hirsh, 1997). A recently developed command shell called Fish Shell includes syntax highlighting and improved auto-completion that allows each command to decide what to auto-complete and shows descriptions of the completions (Fish Shell, 2009). These improvements deal with aspects of command line interaction that are more advanced than our short-term goals for GEKA, so they will not be discussed in detail here, but will be revisited during future design iterations of GEKA.

## **2.4 Literature supporting keyboard interaction**

Several prominent HCI researchers have advocated for text-based interaction methods similar to GEKA.

Gentner and Nielsen (1996) listed many problems with GUIs in colourful language, saying that “direct manipulation quickly becomes repetitive drudgery” and that what they call see-and-point interfaces are “as if we have thrown away a million years of evolution, lost our facility with expressive language, and been reduced to pointing at objects in the immediate environment.” Among their recommendations are a focus on expert users and a more expressive interface focused on language.

Raskin (2000) called for “[an interaction method] that is as fast and physically simple to use as typing a few keystrokes and that makes the commands easier and faster to find than does a menu system.”

Norman (2007) predicted that one of the next “UI breakthroughs” would be related to command lines, stating that “GUIs work well only when the number of alternative items or actions is small.” He cites search (both online search engines and operating system file searches) as an example of where command line type interaction is already prevalent. For example, Google supports some types of commands as search queries – including “1 CAD to USD” to convert currency or “weather Vancouver” to display current weather information. Norman also mentions YubNub (YubNub), a web site that is self described as “a social command line for the web.”

## **2.5 Related applications**

There are several existing applications that have similar functionality to GEKA.

Quicksilver (Blacktree, n.d.), shown in Figure 2-4, is a Mac OS X application that allows many tasks to be completed through the keyboard. All

interactions with Quicksilver begin with a search for an object in its catalog. Basic Quicksilver catalogs only the computer's file system. Plug-ins are available to handle a wide variety of other object types. Objects are selected through an adaptive incremental search mechanism. Once an object is selected, an action can be chosen to execute on the object. For a file, some of the possible actions are "open," "rename," and "move to." Some commands involve a parameter, selected at the end. The full command syntax for Quicksilver is object→action→parameter. This allows for quite a bit of flexibility, especially with plug-in support. For example, free text input can be the object, so using a plug-in that supports email the command "**Hello!**" → **email to** → **Mom** can be executed. Unfortunately, this syntax does not match how a user would typically think of the action. The restriction to only one parameter is also a major limitation.

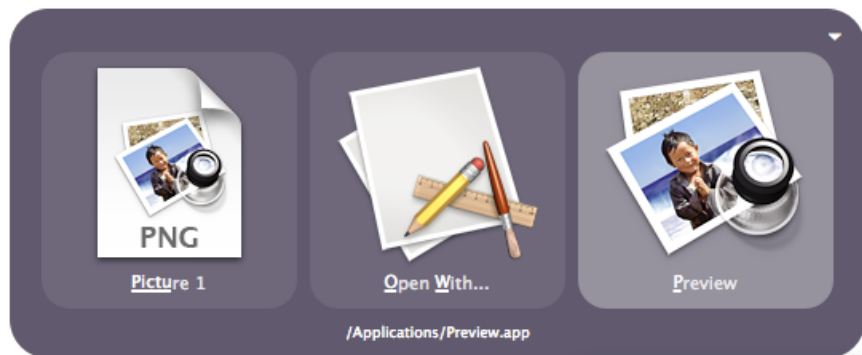


Figure 2-4: Quicksilver with the object "picture 1" command "open with" and parameter "preview" selected.

Enso (Humanized, n.d.), shown in Figure 2-5, is a Windows application that uses text commands for several actions. It has simple syntax: a command name optionally followed by one parameter. Enso often makes use of the current selection in the Windows GUI, for example opening the highlighted file with a specific application or doing a spell check on highlighted text. Enso is available for a small number of commonly used commands such as navigating between windows and looking up words in a dictionary.



**Figure 2-5: Enso with the command “open” and the parameter “firefox” selected.**

Ubiquity (Mozilla Labs, 2009) is a plug-in for Mozilla Firefox, created by a team with the same lead designer as Enso, that supports command input in a way very similar to Enso. It supports commands including translating or mapping addresses in the selected text. A major improvement in Ubiquity over Enso is that it supports multiple parameters, though there is no auto-completion for parameters, and they must be entered in a fixed order.

Inky (Miller, et al., 2008) is described as a “sloppy command line.” It uses a text interface to invoke common browser commands. Many of the challenges of traditional command lines are overcome by including multiple synonyms for command and parameter names and using a very loose syntax.

LAPIS (Miller & Myers, 2000) is a web browser that allows text commands to be entered into the address bar. The commands perform actions on the contents that are displayed in the browser. For example, if the web page contains a table listing information about vehicles, an example command could be “**sort car -by horsepower -order numeric**”. This results in a web page being re-rendered in the browser to display the desired order of cars.

There are several issues with the current generation of applications that

combine keyboard commands and graphical feedback. GEKA attempts to address these issues. Enso, Inky, LAPIS, and Ubiquity have very narrow foci for what their commands can do. Enso focuses on simple text and window manipulation commands and the others focus on the web. Most of these applications have a very limiting syntax as well. Quicksilver and Enso only support single parameters. Ubiquity supports multiple parameters, but they are positional, and thus the user must specify all parameters in a fixed order. Quicksilver uses an adaptive matching algorithm that attempts to move the most likely needed command to the top of the results list based on the user's past behaviour. This can save time, but it means that the user must look at the screen every time a command is typed because there is no way to know which command will match a certain input.

GEKA aims to avoid these problems with three specific improvements over current applications: support for multiple parameters in arbitrary order, smarter matching – including abbreviations for all commands and tolerance for “sloppy typing,” and clear visual feedback of the input characters to facilitate learning and re-use. GEKA's design is described in detail in Chapter 4.

### 3 Formative Study

Our formative study had two distinct phases with the common goal of eliciting requirements for and informing the design of GEKA. We first provide a high level overview of the study goals and then describe each phase in detail before moving on to the results.

Phase 1 was designed to help shape the requirements for GEKA by exploring how advanced users currently interact with WIMP interfaces, specifically with word processors. One of our main foci was to determine how often advanced computer users utilize each of the available WIMP interaction methods, including drop-down menus, context menus, toolbars, keyboard shortcuts, and mnemonics. When this study was conducted, we had not yet made the distinction between toolbar buttons and toolbar drop-downs as described in Chapter 1. Previous work by (Lane, Napier, Peres, & Sándor, 2005) has explored this question, with a focus on keyboard shortcuts. Lane et al. found that keyboard shortcuts were very underused by participants. In fact, keyboard shortcuts were not the most used method for any of the commands that were examined. Our approach in investigating interaction method usage differs from Lane et al.'s primarily in the expertise of our participants. Lane et al.'s participants had significant computer and Microsoft Word experience, but they were mainly professionals who don't necessarily fit our definition of an advanced computer user. Our participants were computer science and electrical and computer engineering graduate students, a category that we believe to be among the most experienced computer users. We suspected that our target users would show more keyboard shortcut usage than did Lane et al.'s. In addition to the focus on more advanced users, we were very interested on qualitative feedback about why participants choose each method. Lane et al. did not collect any such information.

To augment the data that we collected on method usage, Phase 1 of our formative study also looked at user preferences to determine which methods each user favoured and why they might choose to use a method that wasn't their favourite. Finally, we watched users execute a series of commands to see if their actual actions matched their stated preferences and to identify specific problems with current interfaces that GEKA might be able to address.

In Phase 2, we conducted a preliminary evaluation of a GEKA prototype to ensure that we were taking the correct approach and to refine the design of the prototype before continuing with a more rigorous study. The particular prototype design used in the study is not essential to understanding the procedure or most of the results of the formative study, so we will not discuss it here. The full GEKA design process will be described in the next chapter, at which point we will also discuss the qualitative feedback obtained in Phase 2 of the formative study, which is the only data that relies on the specific design. We do this to simplify the presentation of the study and to integrate the one part of the study that deals with design issues with the discussion of the design of GEKA.

The two phases were completed by each participant in a single session.

### **3.1 Phase 1 tasks**

Phase 1 was designed to explore advanced computer users' current behaviour with WIMP interfaces. We chose to work with word processors because they exemplify WIMP interfaces and are complex enough to explore the issues in which we are interested. Participants were given a choice of word processors to work with, based on their experience and preferences. The options were: Microsoft Word 2003, OpenOffice.org 2, and OpenOffice.org 3

for Windows users, and Microsoft Word 2008 for Macintosh users. All participants chose to use Microsoft Word 2003.

In Phase 1, participants were first asked to execute a series of commands the way that they normally would when using Word. This exercise was designed to get each participant to use as many distinct WIMP methods as possible in order to facilitate discussion in a subsequent interview. There were five commands, which were read off by the researcher one at a time. The commands were: **apply bullets**, **align right**, **italic**, **undo**, and **print preview**. We expected most participants to use the toolbar for the first two commands, keyboard shortcuts for the next two, and the drop-down menu for the final command. After executing these commands, participants were asked if they knew of any other interaction methods that they hadn't used for one of the five commands. The methods listed in response to this question and the methods used in the command execution for each participant were explored in detail.

An interview discussing how participants use commands in Word formed the bulk of Phase 1. The specific commands that were discussed were based on Linton, Joy, and Schaefer's (1999) list of the 20 most frequently used word processor commands. The **print default** command was removed because it does not exist in current word processors, and the **spell check** command was removed because it is rarely used explicitly with current word processors. Several common formatting commands were added to give more variety to the discussion. The final list of command is shown in Table 3-1.

**Table 3-1: commands used in the interview portion of our formative study.**

<b>Paste</b>	<b>save</b>	<b>copy</b>
<b>bold</b>	<b>cut</b>	<b>undo</b>
<b>underline</b>	<b>new file</b>	<b>find</b>



<b>superscript</b>	<b>redo</b>	<b>close document</b>
<b>quit</b>	<b>center</b>	<b>Font face</b>
<b>application</b>		
<b>font size</b>	<b>apply style</b>	<b>font colour</b>
<b>zoom</b>	<b>open</b>	<b>print</b>
<b>Save as</b>	<b>print preview</b>	<b>header</b>
<b>insert table</b>	<b>insert picture</b>	

For each of these commands, participants were asked how frequently they use the command (frequently, sometimes, rarely, or never), which method (drop-down menu, context menu, toolbar, keyboard shortcut, or mnemonic) they most frequently use for the command, which other methods they sometimes use, and which methods they know but never use.

The next step was to determine participants' method preferences as well as the magnitude of their preferences. They were given small pieces of paper representing each of the WIMP methods that they had said they use and instructed to place each of these along a meter stick with 0cm being the most pleasant method they can imagine and 100cm being the least.

After rating the methods, each one was discussed in detail. Participants were asked what they like and dislike about each method as well as the reasons for not always using their most favourite method and the reasons for resorting to each of the others.

For the final part of Phase 1, participants were asked to complete another series of commands using Word. These commands all involved parameters. They were chosen in order to see how users interact with dialog boxes. The commands are listed in Table 3-2.

**Table 3-2: Commands used in dialog box exploration during our formative study.**

<b>Print 5 copies</b>	<b>print page 2 in landscape format</b>
<b>print 10 copies of page 2</b>	<b>replace all occurrences of "the" with "an"</b>
<b>set line spacing to 1.5</b>	<b>save the document as "new file" in a folder called "save as."</b>

It was noted how participants completed each command: completely with the mouse, completely with the keyboard, or started with the keyboard and then finished with the mouse. After all of the commands were completed, participants were asked to discuss why they used a particular combination of methods.

### **3.2 Phase 2 tasks**

Phase 2 was designed as a preliminary evaluation for an early GEKA prototype running in OpenOffice.org Writer 2. Participants were given a demonstration of GEKA and then asked to complete a series of commands one at a time to practice using GEKA. The commands are listed in Table 3-3.

**Table 3-3: Commands used in GEKA practice during Phase 2 of our formative study.**

<b>bold</b>	<b>redo</b>
<b>cut</b>	<b>set font to Arial</b>
<b>undo</b>	<b>set font size to 25</b>
<b>underline</b>	<b>set background colour to black</b>

<b>paste</b>	<b>insert table with 5 rows and 3 columns</b>
<b>center</b>	<b>zoom to 200%</b>
<b>copy</b>	<b>print 5 copies</b>
<b>superscript</b>	<b>print 10 copies of page 2</b>
<b>print preview</b>	<b>print page 3 in landscape format</b>
<b>close document</b>	<b>set line spacing to 1.5</b>

After these practice commands, participants were given a guided task to perform. A document with no formatting was opened on the computer, and they were given a printout showing the desired result, which included formatting and some textual changes. The printout was annotated where the desired result was ambiguous, for example, font sizes were labeled. Participants were asked to apply the formatting and to make the text changes. They were asked to use GEKA as frequently as possible when executing commands. The complete formatted and unformatted documents are shown in Appendix A. The commands used in completing the task are listed in Table 3-4.

**Table 3-4: Commands used during the guided task in Phase 2 of our formative study.**

<b>right align</b>	<b>insert date</b>	<b>insert page numbers</b>
<b>center align</b>	<b>font size</b>	<b>underline</b>
<b>highlight</b>	<b>font colour</b>	<b>bullets</b>
<b>superscript</b>	<b>strikethrough</b>	<b>insert hyperlink</b>
<b>insert table</b>	<b>subscript</b>	

All comments made by participants and any apparent difficulties seen during the task were noted. After the task, participants were asked general interview questions about their experience and then were asked a series of Likert scale questions comparing GEKA to each of the WIMP methods that they used in Phase 1, as well as to dialog box navigation with mouse and with keyboard. These Likert scale questions asked participants to compare WIMP and GEKA in terms of speed, ease of learning, ease of remembering, ease of use, and overall preference.

### **3.3 Apparatus**

All tasks involving computer use were conducted on an Apple MacBook laptop running Windows XP in a dual boot setup using Apple's Boot Camp software with a 13" screen at a resolution of 1280x800. An external mouse and Windows keyboard were used. Phase 1 tasks were completed using a standard installation of Microsoft Word 2003 without any customization. Phase 2 tasks were completed using OpenOffice.org Writer 2 with a GEKA prototype running as a Java plug in.

### **3.4 Participants**

Because GEKA is designed primarily as a tool to help advanced computer users, this study was interested only in advanced users. Recruiting was done through an email sent to the UBC computer science graduate student mailing list. We had 10 participants (3 females), of which nine were computer science graduate students, and one was an electrical and computer engineering graduate student. Participants were compensated with \$15 for their time.

### **3.5 Procedure**

Each participant completed a single 90-minute session. After a brief introduction to the study, and completing a questionnaire gauging their computer and word processor experience, participants completed all of Phase 1 followed by all of Phase 2, as described above.

### **3.6 Results**

We describe the data analysis and results for the two phases of the formative study.

#### **3.6.1 Phase 1**

As expected, we found a very low rate of mnemonic usage, with only two participants reporting using mnemonics at all, and then only for a very small number of commands. Because of this, mnemonics are left out of our analysis.

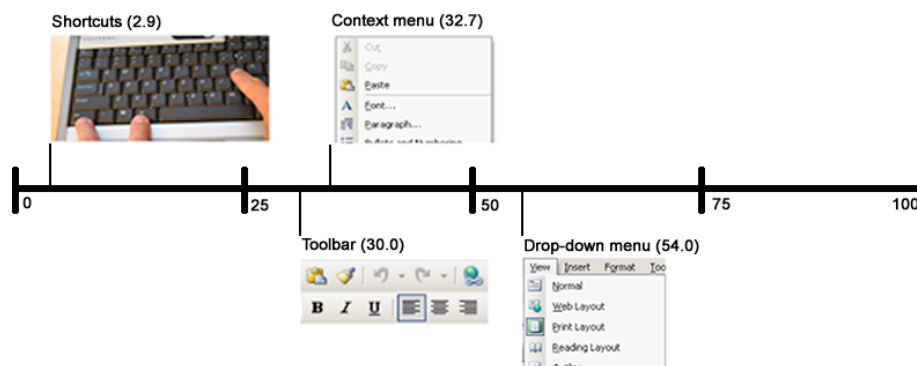
Table 3-5 shows the reported command usage from the interview portion of Phase 1. Of the 10 participants, there were only two cases where anyone indicated that they didn't use a command. Far more commands were reported to be frequently- or sometimes-used than rarely-used, so we conclude that we chose a command set that is fairly representative of what our participants generally use.

We were expecting to find significantly more shortcut usage with our participants than (Lane, Napier, Peres, & Sándor, 2005) did with their less experienced participants, and indeed we did. For 11 of our commands, at least half of our participants indicated using keyboard shortcuts more often than any other method. While this is much higher usage than Lane et al. found, we still consider it to be fairly low, especially considering the preferences that our participants stated.

**Table 3-5: Command usage in Word 2003.** The second through fifth columns show the number of participants who use each command with each frequency. The sixth column shows the number of participants who know shortcuts for each command. The final four columns show the number of participants who use each method most frequently for each command. The top row shows which percentage of commands fall into each category.

	Frequently	Sometimes	Rarely	Never	Know kb shortcut	Menu	Shortcut	Context Menu	Toolbar
Percent of cmds	48%	31%	20%	0%	46%	25%	39%	0%	36%
Paste	9	1	0	0	10	0	10	0	0
Save	9	1	0	0	10	0	9	0	1
Copy	9	1	0	0	10	0	10	0	0
Bold	6	3	1	0	10	0	9	0	2
Cut	7	2	1	0	9	0	8	2	0
Undo	7	3	0	0	9	2	9	0	0
Underline	5	2	3	0	10	0	7	0	4
New File	7	1	2	0	7	3	6	0	2
Find	6	4	0	0	9	2	8	0	1
Superscript	0	5	5	0	5	4	5	1	0
Redo	4	2	4	0	5	3	5	0	2
Close Doc	6	3	1	0	4	1	3	0	7
Quit App	8	1	1	0	4	0	3	0	8
Center	3	4	3	0	1	0	1	0	9
Font	7	2	1	0	1	0	0	0	10
Size	7	3	0	0	2	0	2	0	8
Style	1	6	3	0	0	1	0	0	9
Font Color	1	4	5	0	0	0	0	0	10
Zoom	3	4	3	0	0	1	0	0	9
Open	6	3	1	0	8	5	3	0	3
Print	6	4	0	0	5	8	2	0	1
Save as	5	3	2	0	1	8	1	0	0
Print Pre	4	4	2	0	0	7	0	0	2
Head	0	4	6	0	0	8	0	0	1
Table	0	6	3	1	0	6	0	0	3
Insert Picture	0	5	4	1	0	7	0	0	1

Figure 3-1 shows the average rating that participants gave each WIMP technique, with 0 being the best score, and 100 being the worst. All 10 participants rated keyboard shortcuts as their most preferred method and drop-down menus as their least preferred method. Toolbars and context menus varied between second and third place. The positioning on the scale indicates that the preference for keyboard shortcuts is quite strong, with a rating very close to zero. Similarly, the dislike for drop-down menus appears strong. They are rated nearly twice as bad as context menus and toolbars. Common reasons for liking keyboard shortcuts included speed, precision, and being able to keep one's hands in the same place. Similarly, common reasons for disliking menus included the need for multiple clicks and scanning through the options to find the right one. Seven of the 10 participants stated that they generally prefer using the keyboard during word processing because their hands are usually already there.

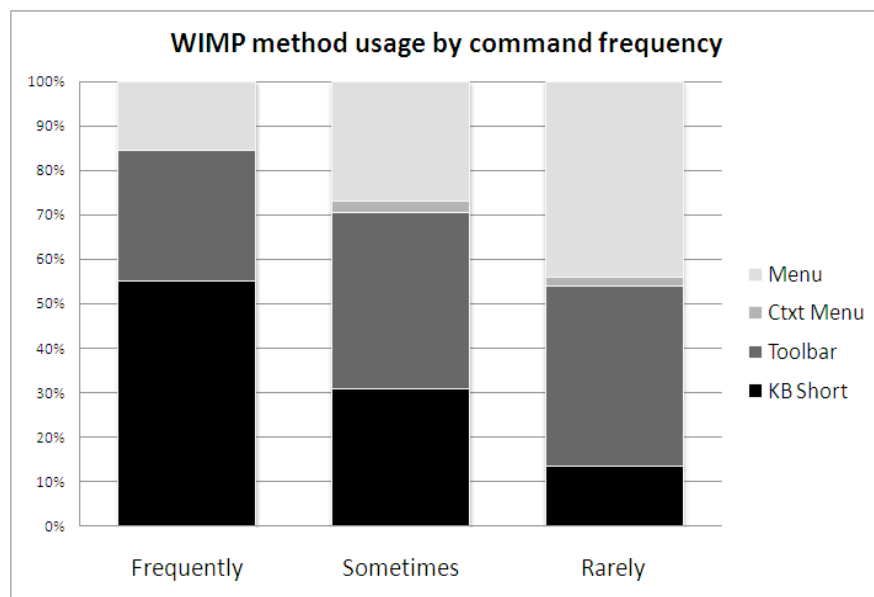


**Figure 3-1: Preferences for each WIMP method. A rating 0 is best, and 100 is worst (N=10).**

These preferences, examined in conjunction with the usage data, indicate significant problems with current WIMP interfaces. The average participant uses keyboard shortcuts most frequently for 10 of the 26 commands that we examined, the toolbar for 9 commands, and menus for 7 commands. While keyboard shortcuts are used more often than each of the

other methods, grouping the methods into keyboard versus mouse paints a different picture. Only 10 commands use the keyboard but 16 use the mouse, even though keyboard shortcuts were strongly preferred over mouse-based methods. Of the 7 commands where drop-down menus were the most used method, 3 were commands that most participants reported using frequently, suggesting that our participants often resort to their least favourite interaction method to execute commands.

Figure 3-2 sheds further light on this issue by breaking down method usage by frequency of command usage. Keyboard shortcuts are in fact used for the majority of frequently executed commands, but this is a rather slim majority; for sometimes- and rarely-used commands, mouse-based methods dominate.



**Figure 3-2: Percentage of commands reported to be most frequently executed, with each technique broken down by frequency of command use ( $N=10$ ).**

The most common reason that participants gave for not using keyboard shortcuts was simply that they did not know the shortcut. This could be either because a shortcut does not exist for a command, or because learning and remembering the shortcut is too difficult. Indeed, 5 of the commands we



discussed do not have keyboard shortcuts in Word 2003, and of the remaining 21 commands, participants on average only knew 12 shortcuts.

Table 3-6 shows the breakdown of keyboard and mouse usage in the six commands involving parameters that participants executed. Each row shows a specific combination of methods used to open the dialog box and then to navigate within the dialog box, with the first column showing whether the dialog box was opened with a keyboard shortcut or a mouse-based method, and the second column showing whether the keyboard, the mouse, or a combination of the two was used to navigate through the fields in the dialog box. The penultimate row, “other,” shows two cases where the line spacing was set through the toolbar, so no dialog box was used.

**Table 3-6: Keyboard versus mouse usage for the six command executions involving parameters. The final column shows the total number of commands executed using the combination of methods indicated in the first two columns (N=10).**

Dialog opened with	Values selected with	Count
Keyboard	Keyboard	6
Keyboard	Mouse	5
Keyboard	Both	5
Mouse	Keyboard	0
Mouse	Mouse	40
Mouse	Both	2
Other		2
Total		60

In the majority of cases, 40 of the 58 times dialog boxes were used, the mouse was used both to open and to navigate the dialog box. This is perhaps not surprising considering that two of the six commands that were used do not have keyboard shortcuts and that participants indicated that they generally used the menu to select the print command. When a command that invokes a dialog box is selected using the mouse, it is not unreasonable to continue using the mouse for parameter selection. What is surprising is looking at the 16 cases where a keyboard shortcut was used to select the command. In 10 of those cases, the mouse was used either exclusively or in conjunction with the

keyboard to select parameters. Thus, even when a dialog box is opened with the keyboard and participants have indicated a general preference for the keyboard, the mouse ended up being used 62.5% of the time in order to finish the dialog box for the command. When participants were asked why they would use the mouse in a dialog box, the overwhelming answer was that the **TAB** key navigation through dialog boxes is slow and confusing because the cursor jumps around the dialog box in a way that seems completely unpredictable and random.

### **3.6.2 Phase 2**

A full understanding the qualitative feedback for the GEKA prototype requires an understanding of the prototype design, so that will be discussed in the next chapter after the design description. This section will focus only on the Likert scale responses shown in Figure 3-3.

Figure 3-3: Likert scale responses from Phase 2 of the formative study. The 17 situations where most participants favoured GEKA are shown in green (lighter shading), while the 17 where most participants preferred the WIMP method are shown in red (darker shading). Those with no clear preference are in white. Some methods do not have ten responses, either because a participant did not indicate having used that method or because they didn't feel they had enough GEKA experience to compare to it (N=10).

Keyboard Shortcuts			
GEKA is:	Better	Same	Worse
Speed	0	0	10
Learn	7	2	1
Remember	8	2	0
Ease of use	2	3	5
Overall	2	2	6

Toolbars			
GEKA is:	Better	Same	Worse
Speed	5	2	2
Learn	6	0	3
Remember	3	2	4
Ease of use	3	3	3
Overall	6	2	1

Menus			
GEKA is:	Better	Same	Worse
Speed	10	0	0
Learn	6	2	2
Remember	7	3	0
Ease of use	8	1	1
Overall	9	1	0

Context Menus			
GEKA is:	Better	Same	Worse
Speed	5	1	1
Learn	2	4	1
Remember	3	2	2
Ease of use	3	2	2
Overall	3	2	1

Dialog Box with Mouse			
GEKA is:	Better	Same	Worse
Speed	8	1	1
Learn	1	4	4
Remember	0	5	5
Ease of use	6	2	2
Overall	6	1	1

Dialog Box with Keyboard			
GEKA is:	Better	Same	Worse
Speed	6	1	1
Learn	3	5	0
Remember	4	3	1
Ease of use	5	2	1
Overall	7	1	0

Participants unanimously felt that GEKA was slower than keyboard shortcuts, and they generally preferred keyboard shortcuts. This is not surprising because GEKA does in fact require more keystrokes than do shortcuts. However, participants did find GEKA to be easier to learn and remember than shortcuts, which is a major advantage considering that the primary reason for not using shortcuts was difficulty in learning and remembering them. Participants felt that GEKA was faster than all of the mouse-based WIMP techniques and preferred it over all of them except for context menus, which were considered the same (i.e., a neutral rating). Participants also found GEKA faster and easier than dialog boxes, both with the mouse and with the keyboard. This is another major success for GEKA because as we have already shown, participants often have a very difficult time using current dialog boxes quickly and effectively.

These responses, while very preliminary and qualitative, suggest that our goals for GEKA are on the right track to becoming a useful augmentation to existing user interfaces.

### **3.7 Limitations**

The command usage data in Phase 1 is imperfect because it was collected through self-report rather than actual usage data. There should be no reason for participants to be dishonest about usage, but it is often very difficult to reflect on previous actions and report them in this way. Thus, despite the participants' best intentions, it is possible that the numbers do not fully reflect actual usage.

All of the data reported from Phase 2 was based on self-report and thus could be biased by participants' desire to please the researchers. This was less likely to be a problem in Phase 1 because participants would not have known what answers the researchers might be hoping to get.

### **3.8 Discussion and conclusions**

Phase 1 of our formative study showed that current WIMP interfaces are not meeting the needs of advanced computer users. In at least the context of word processing, advanced users prefer to execute commands through the keyboard. In many cases, there are a number of hurdles preventing this from being possible. Keyboard shortcuts are often not available for commands, and when they are, even advanced computer users have trouble learning and remembering all of the shortcuts for commands that they use. When a command has parameters, a dialog box is generally used. While it is possible to navigate a dialog box with the keyboard, even advanced users rarely do so apparently because the current navigation methods are very slow and confusing.

These findings indicate a great opportunity for GEKA. If GEKA can avoid the drawbacks of keyboard shortcuts and dialog boxes by being available for most commands, being easy to learn and remember, and handling parameters in a straightforward way, the participants that we studied would almost certainly be glad to have GEKA available and make frequent use of it.

Phase 2 showed that an early GEKA prototype was on the right track to meeting those goals, with most participants feeling that it was easier to learn and remember than shortcuts, and that it was faster than all mouse-based WIMP methods and all dialog box methods.

With the information that we learned in this formative study, we were able to make improvements to the GEKA design as described in the next chapter and proceed to a laboratory experiment, described in Chapter 5, to more thoroughly compare GEKA to existing WIMP techniques.

## **4 GEKA Design and Prototypes**

We designed and implemented two GEKA prototypes, one for the formative study described in the previous chapter, and one for the laboratory study described in the next chapter. The design of these prototypes and the goals that helped shape the design process are described here.

### **4.1 Goals**

Our prototype design was motivated by a set of concrete design goals as well as our general vision for GEKA and our long term goals. We describe each of these.

#### **4.1.1 GEKA vision**

The highest level goal for GEKA is to make computing more pleasant and productive for advanced users. This could come from increased speed, reduced errors, reduced cognitive demand, or simply preference, and will be achieved by allowing most commands in most applications to be executed through the keyboard. We see GEKA as an addition to WIMP interfaces, augmenting them with an additional input mechanism rather than removing or replacing any existing features. As such, the additional choice in interaction methods is strictly a benefit: it can be utilized when users see fit; the rest of the time, it should not detract in any way from their experience.

We envision a future version of GEKA that can be used in a consistent manner across all applications. While our initial prototypes are designed for the command set of a specific application domain, we tried to not make any decisions that would limit GEKA's usefulness in other domains.

### 4.1.2 Prototype goals

The following list summarizes the concrete goals that our prototypes were designed to meet. They are based on our vision for GEKA as well as the results from Phase 1 of the formative study discussed in the previous chapter.

*Speed* – In the formative study, all participants identified keyboard shortcuts as their favourite interaction method. The primary reason for doing so was speed. Clearly speed of execution of commands is very important to advanced users. In order to be successful, GEKA must be fast.

Because we decided that GEKA would not replace any current WIMP features, keyboard shortcuts will continue to be available alongside GEKA. We expect continued use of keyboard shortcuts when they are available and known to users, so it is not necessary that GEKA match their speed. What is important is that GEKA be noticeably faster than the mouse-based WIMP techniques that our formative study participants wanted to avoid using.

*Error rate* – It is important for any interaction method to have a low error rate. A constant need to identify and correct errors while executing commands is frustrating and time consuming. Error rate did not come up as an issue during our discussion of WIMP methods in the formative study, so we can conclude that users are satisfied, or at least not overly dissatisfied, with their error rates using both keyboard shortcuts and mouse-based methods. While outperforming WIMP would certainly be desirable, we feel that simply matching the error rates of WIMP interfaces is enough for GEKA to be considered a success on this measure.

*Ease of learning and remembering commands* – The biggest drawback that we found for keyboard shortcuts was that participants often simply did not know them, which was due in large part to difficulty in learning and remembering shortcuts. In order to be a viable alternative to mouse-based WIMP methods, GEKA must be easier to learn and remember than keyboard shortcuts, and ideally as easy as menus and toolbars. We are hoping to

accomplish this by making extensive use of graphical feedback so that GEKA interaction is based on recognition rather than recall.

*Visual demand* – High visual demand is a major drawback of pointer based interaction methods. Users often must scan through long menus or many toolbar items to find the desired command. Even when the exact location of the command is known, the user's visual attention must be completely focused on the pointer, ensuring that it is in the proper location before selecting the item. This can distract users heavily from their main task, sometimes causing significant frustration and time loss. GEKA is designed in a way that allows advanced users to execute commands with little or no visual attention.

Note that there is a potential conflict between this goal and the previous goal, which requires the use of visual feedback. GEKA must include visual feedback that is readily available when needed but does not distract more experienced users who do not rely on it.

*Completeness* – GEKA must be designed in a way that will support most commands and parameters. This, along with ease of learning and remembering, should address users' main issue with keyboard shortcuts: that there are too many commands for which keyboard shortcuts are unknown or simply don't exist. Our prototypes do not aim to implement the full command set for their application domain, but the command structure is designed to place as few restrictions as possible on the types of commands that can be implemented so that a complete set could eventually be implemented.

#### **4.1.3 Long term goals**

We can already anticipate some features that might be added to future GEKA iterations. While the prototypes were not specifically designed to accommodate any of these features, they were kept in mind throughout the process. We tried to not make any decisions that would adversely impact our ability to implement them in the future. These features include command and



parameter history, aliasing, scripting, and inter-application communication. They will be discussed more thoroughly in the final chapter of this thesis.

## **4.2 Prototype design**

We designed and implemented a GEKA prototype to use in Phase 2 of our formative study (previous chapter) and another for our laboratory experiment (next chapter). Consistent with Phase 1 of the formative study, we chose to implement the prototype in the domain of word processing because word processors have rich, complex feature sets appropriate for studying, word processor interfaces are generally very typical WIMP interfaces, and most computer users are familiar with word processors.

The prototypes used in each of the two studies differed in two ways. The formative study prototype ran as a plug-in within OpenOffice.org Writer in order to give participants the true experience of using GEKA in a fully functional word processor, whereas the laboratory experiment used a replica of the Microsoft Word 2003 user interface but with a very limited command set implemented. In addition, some slight changes were made after the formative study based on participant's feedback. Despite these differences, the appearance and functionality of the GEKA design was very similar in both versions, and the implementation was based on the same source code. The two versions will be described as one, with the changes that occurred as a result of the formative study being described when appropriate.

There are two major components to the GEKA design. The first is the command language, including the actual command structure as well as the auto-completion mechanism. The second is the graphical feedback, which provides cues and shows command and parameter options. Each component will be described separately.

### **4.2.1 GEKA command language**

The core of GEKA is its command language. GEKA commands need to be flexible and expressive while both keeping keystrokes low and being easy to learn and remember. The flexibility and expressiveness come from a command syntax that closely resembles those of powerful traditional command line interfaces. The reduction of keystrokes and ease of learning and remembering commands are provided by a uniform auto-completion mechanism for both command names and parameters that lists results effectively, shifting the focus of interaction to recognition rather than recall.

#### **4.2.1.1 Command syntax**

The GEKA command syntax relies on existing application-specific command and parameter names that users should already be familiar with through their WIMP interactions with an application. This should greatly reduce the time and difficulty of learning commands for experienced users, potentially eliminating it entirely in some cases. The downside of this is that command and parameter names are often quite long. This is addressed by the auto-completion mechanism described below, and by giving each command a short name in addition to its full name. The short name is a sequence of a few characters that will uniquely select a command or parameter.

GEKA uses a prefix command syntax with keyword parameters. This command structure provides a straightforward and flexible way to execute commands and will be familiar to users who have experience with CLIs. For most GEKA commands with parameters, a command name is selected, and then parameters may be selected by specifying a parameter name and then a value. There is one exception: for commands with only one parameter, the parameter name is left out, leaving just the command name and the parameter value. Examples of GEKA commands include:

**Table 4-1: Examples of GEKA commands.**

<b>bold</b>	<i>Command with no parameters</i>
<b>zoom 200</b>	<i>Command with one parameter</i>
<b>print copies 4</b>	<i>Command with multiple parameters, where only one parameter is used</i>
<b>Insert_table rows 4 columns 2</b>	<i>Command with multiple parameters used</i>

#### **4.2.1.2 Auto-completion**

The auto-completion mechanism is what gives GEKA its ease of learning and remembering commands and parameters and its ability to require only a small number of keystrokes. After each character that the user types, an incremental search mechanism finds all of the commands or parameters that the user may have intended. The search results are presented in a graphical match list whose ordering is intended to approximate the likeliness that each was intended. This eliminates the need to memorize and type full names.

To order the match list, matches are divided into four categories, described below:

*Exact match* – An exact match is a command or parameter whose name (either the full name or the short name) is exactly the text that has been entered. In order for this to work, names and short names must be designed in such a way that all short names are unique and the short name for one command is not the full name for another command.

*Prefix match* – A prefix match is a command or parameter whose name begins with the characters in the entered text. For example,

the commands **save**, **save as**, and **save as web page** are all prefix matches for the input “sav”.

*Substring match* – A substring match is a command or parameter that contains the characters in the entered text as a contiguous sequence. For example, the commands **superscript**, **subscript**, and **full screen** are all substring matches for the input “scr”.

*Subsequence match* – A subsequence match is a command or parameter that contains all of the characters in the entered text in order, though there may be other characters in between. For example, the commands **background colour**, **font colour**, and **clear** are all subsequence matches for the input “clr”.

The results within each category are ordered lexicographically, and all matches within one category are displayed before any of the matches from the next category.

We considered using an adaptive mechanism to order the matches based on the user’s history with executing GEKA commands. This could potentially reduce the number of keystrokes needed to execute a command by bringing frequently used commands to the top of the list more quickly. However, there would be a major drawback in that the match list would be unpredictable and users would be required to look at the feedback to determine which command is selected after inputting text. Thus, we decided that a static sorting algorithm was more consistent with our goal of low visual attention because once a user learns a character sequence that will bring the desired result to the top of the list, the pairing will never change, and if the sequence is remembered, the command or parameter can then be used without looking at the screen.

The match ordering algorithm is based on our intuition about how users would search for commands. This assumption should be validated in a future study.

#### **4.2.2 GEKA graphical feedback**

The second part of the GEKA design is the graphical feedback that displays the auto-completion match lists and all of the other information needed to make GEKA interaction quick and easy. We designed and implemented a simple graphical feedback component in order to test the command language as described above. In contrast to the command language, which we believe to be robust and complete, this initial graphical feedback component is less complete. We plan to iterate and improve upon it in future work.

The primary purposes of the graphical feedback component are to prompt the user to enter characters and display the characters that have been typed, list the possible options for command and parameter names, refine the lists according to the matching algorithm above as text is entered, and confirm the selected command and parameters so that the user is confident the correct combination will be executed.

The graphical feedback component performs these core functions in a very straightforward way, borrowing heavily from the design of Quicksilver (Blacktree), which has proven itself to be well suited to this type of keyboard interaction. The main purpose of the graphical feedback prototype is to be able to test the command language with users, determining whether our core approach to the problem is on the right track. Our formative study and our laboratory experiment were designed to explore issues including how easily users can learn to use GEKA, how their performance compares to WIMP performance after practice, and how much they like using GEKA. All of these can be addressed with a simple, straightforward feedback component. We were not concerned with our initial prototypes being attractive or flashy,

making novel use of visual information, or necessarily supporting all of the commands that are possible within the GEKA command language.

We will give an overview of how the graphical feedback combines with the command language to allow the user to execute commands and then describe each part of the graphical feedback in detail.

#### **4.2.2.1 Overview of GEKA interaction**

GEKA exists as a separate mode within the WIMP application. It is entered through the key combination **CTRL+ENTER**, which brings the GEKA window to the front of the screen. Figure 4-1 shows an overview of the GEKA graphical feedback component.

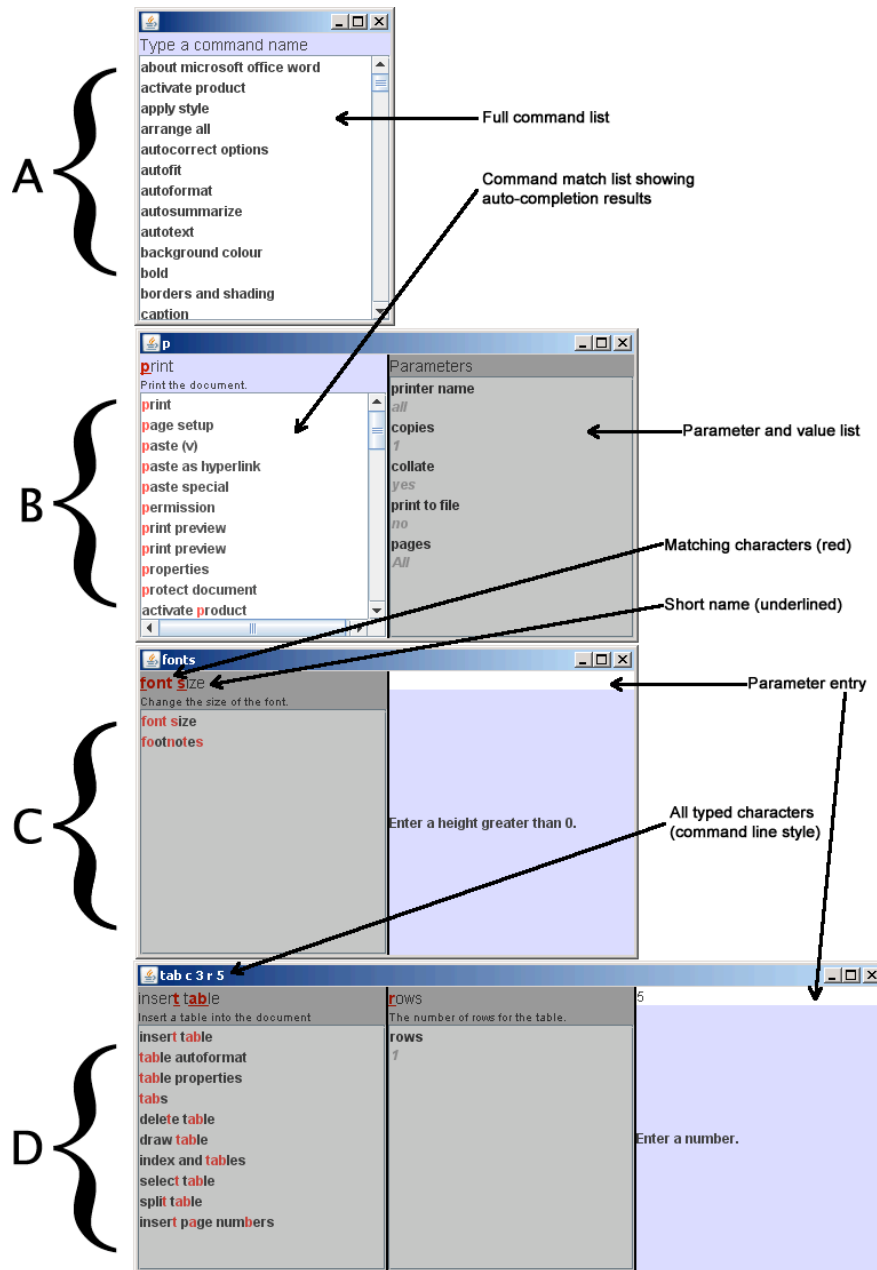


Figure 4-1: GEKA prototype graphical feedback showing four distinct stages of interaction. Stage A is before anything has been typed, B shows command name selection, C shows inputting a parameter for a command with only one parameter, and D shows inputting a parameter for a command with multiple parameters. In each stage, the blue and white pane has the input focus.

Initially, a list of all possible commands is shown with a prompt for the user to begin typing, as seen in Part A of Figure 4-1. As the user types, the command list is refined to include only the match list results as described in the command language section. When the best-match command has parameters, a second pane listing the parameters and their current values is shown on the right, as seen in Part B.

When the selected command has parameters, pressing **SPACE** will move the focus (shown by displaying the focused panel in blue and white while the other panels are grayed out) to the second panel, allowing parameter name selection in a manner identical to command selection. After a parameter name is selected, pressing **SPACE** again will open and move focus to the parameter value pane, as shown in Part D. When the value is selected, pressing **SPACE** another time will move focus back to parameter name selection, and any number of further parameter name and value pairs can be selected. Pressing **ENTER** at any time that a valid command and set of parameters are selected will execute the selected command with the parameter values, if applicable. Pressing **ESCAPE** will close the GEKA window without executing a command.

Parameter selection differs slightly if the command has only one parameter. In this case, there is no need to select the parameter name. The first press of **SPACE** will shift the focus straight to value selection, as shown in Part C.

The title bar of the GEKA window shows all characters that have been entered. It resembles a traditional command line. In the command or parameter name at the top of each pane, the characters that match the input are shown in red, and the characters in the short name for the command or parameter are underlined.

The use of **SPACE** to move between panes in GEKA came out of the formative study. Initially, we had been using **TAB** for this purpose, because



**TAB** is traditionally used to move between parts of a GUI. Many of our participants reported that this was confusing for two reasons. First, after a parameter value is selected, the focus actually moves backward, and the **TAB** key moving forward sometimes and backward sometimes felt very awkward. Second, it did not feel like using a command line interface in which **SPACE** is typically used to denote the end of a command or parameter.

Another improvement made based on the formative study is the use of **BACKSPACE** to remove characters from the input. Initially, there was no way to “undo” a parameter value that had already been set. The way to correct a mistake was simply to input the value a second time. Some of our participants intuitively tried to use **BACKSPACE** for this purpose, so we added this functionality. Pressing **BACKSPACE** at any time will remove the most recent character from the input. In some cases, this is very straightforward. For example, if a parameter value is currently being set and “abc” has been entered; pressing **BACKSPACE** will simply remove the ‘c’ leaving “ab.” It becomes more complicated when eliminating a character would change the focus of the graphical feedback. For example, in Figure 4-2, the focus is on parameter value entry, and the input is “p c ” with the last character being a **SPACE**. Pressing **BACKSPACE** at this point would eliminate the **SPACE** from the input. Because that **SPACE** was what triggered the move to the parameter value entry, and it is now no longer part of the input, the focus will move back to parameter value selection with the input string “p c” as shown in Figure 4-3.

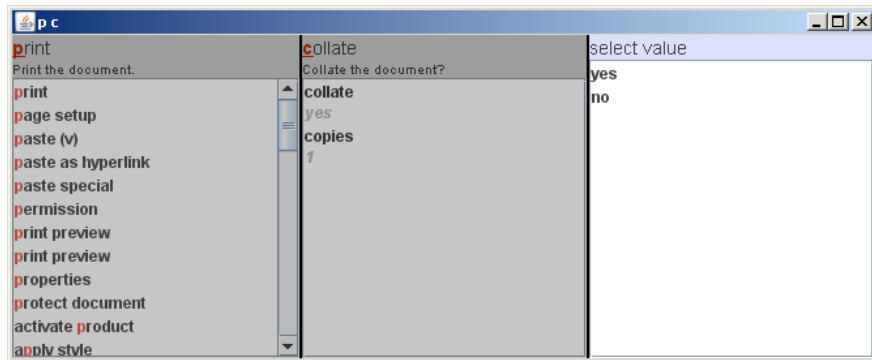


Figure 4-2: GEKA prototype after the characters “p c ” have been entered. There is a trailing SPACE after the ‘c’ that opened the parameter value pane.

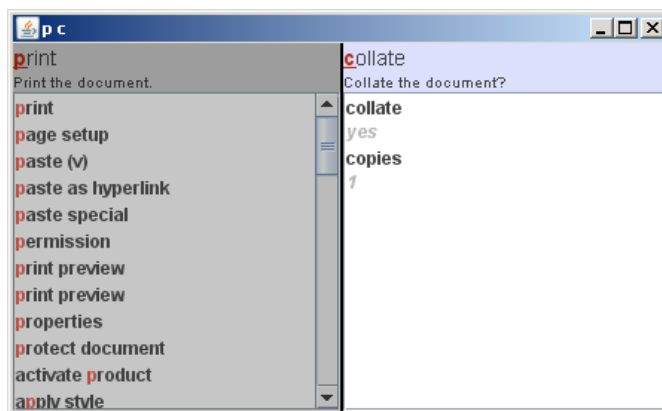


Figure 4-3: The result of pressing BACKSPACE in Figure 4-2, removing the trailing SPACE from the input and leaving just the characters “p c”.

We next describe each aspect of the graphical feedback in detail.

#### 4.2.2.2 Command list

The command list is shown in a pane that lists each command on a separate line. Before any text has been entered, all possible commands are listed, and a prompt is displayed at the top asking the user to type a command name. This is illustrated in Figure 4-4.

As characters are typed, the list of commands updates to show only those commands that match the input according the matching algorithm. The prompt to type a command is replaced with the first command in the match

list, indicating that if the enter key is pressed, that command will be executed. A short description of each command is also displayed to remind the user of its functionality. Figure 4-5 shows GEKA after one character, 's' has been typed. The command **save** has 's' as its short name and is brought to the top of the list. Figure 4-6 shows just the command list after the characters "tab" have been entered. This list contains all four match categories, and they are labeled in the figure.

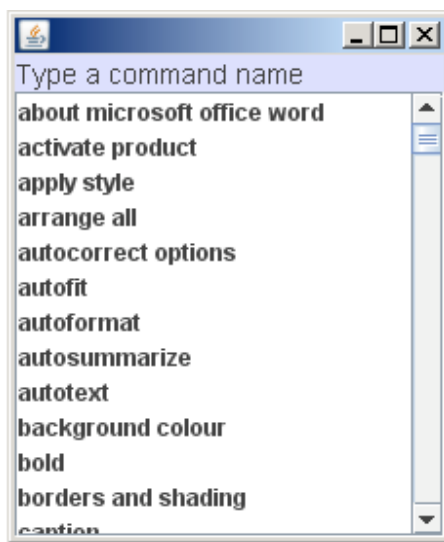


Figure 4-4: Initial GEKA command list with all possible commands. The user is prompted to type a command name.

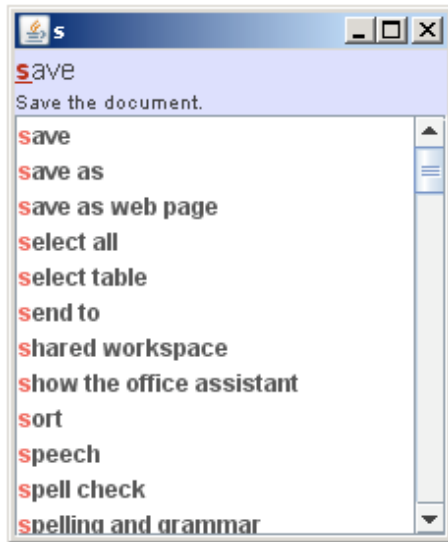


Figure 4-5: GEKA command list after the character 's' has been typed. The save command is shown at the top because it is the best match. It will be executed if ENTER is pressed.

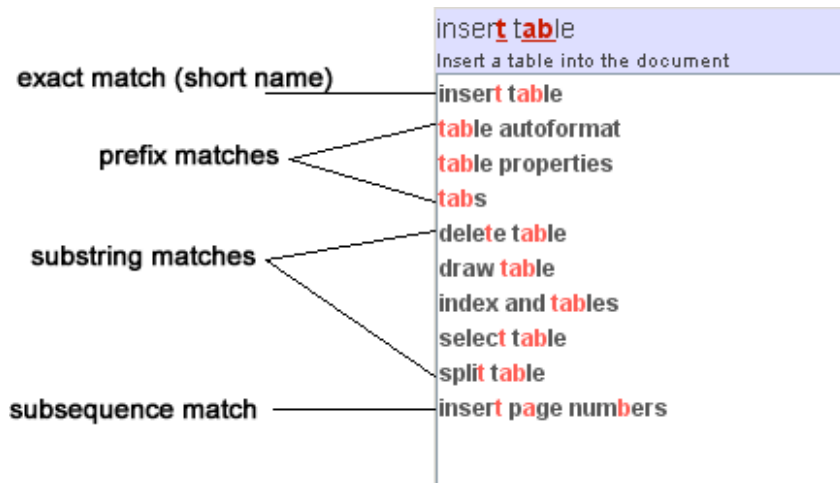


Figure 4-6: GEKA command list after the characters 'tab' have been typed. The best matching command is insert table. This command list contains all four categories of match described in the command language section.

The down and up arrows can be used to scroll through the command list. Figure 4-7 shows GEKA after “align” has been entered. If the user was looking for the command **left alignment**, it would not be possible to bring that command to the top based on the matching algorithm. However, the

command can be selected by using the down arrow. Figure 4-8 shows GEKA after **left alignment** has been selected by entering the text “align” and then pressing the down arrow twice.

If a letter, number, or the **BACKSPACE** key is typed, any down and up arrow input is disregarded, and the match list is updated according to the auto-completion algorithm with the first result in the list becoming the current selected command.

In both the command list and the best match command at the top, the characters that match the input string are shown in red. In the best match command at the top, the characters comprising the command’s short name are shown underlined.

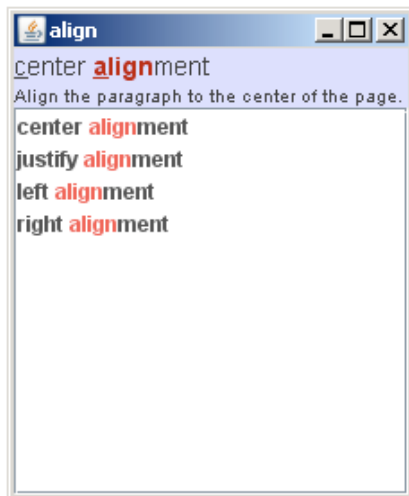


Figure 4-7: GEKA command list after the characters 'align' have been entered. There is no way to select a command other than center alignment by continuing to type characters.

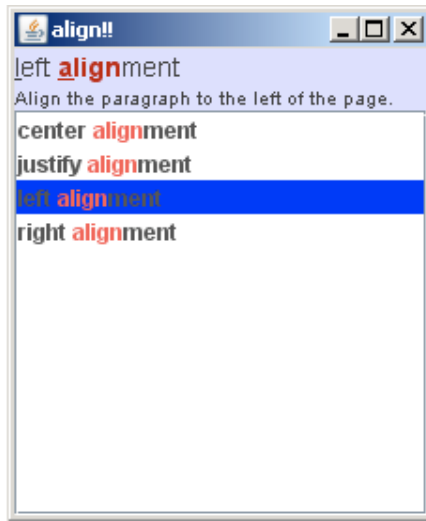


Figure 4-8: GEKA command list after the characters 'align' have been entered and the down arrow has been pressed twice, moving the selection to the command left alignment. The text entry area, discussed in section 4.2.2.5 displays exclamation marks (!) for the down arrows.

#### 4.2.2.3 *Parameter name list*

The parameter list looks and acts very similarly to the command list. The only difference is that in addition to the parameter name, the current value of the parameter is listed. The value is shown on a separate line and in a different colour to make it distinct from the parameter name. Figure 4-9 shows the parameter list for the command **insert table**. Figure 4-10 shows the same list after the character 'c' has been typed.

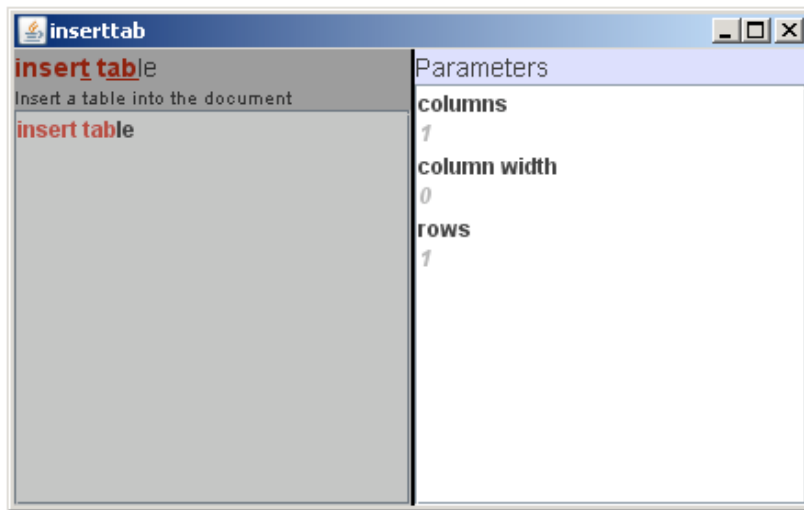


Figure 4-9: GEKA window with the parameter list in focus. This parameter list shows all parameters for the command 'insert table'

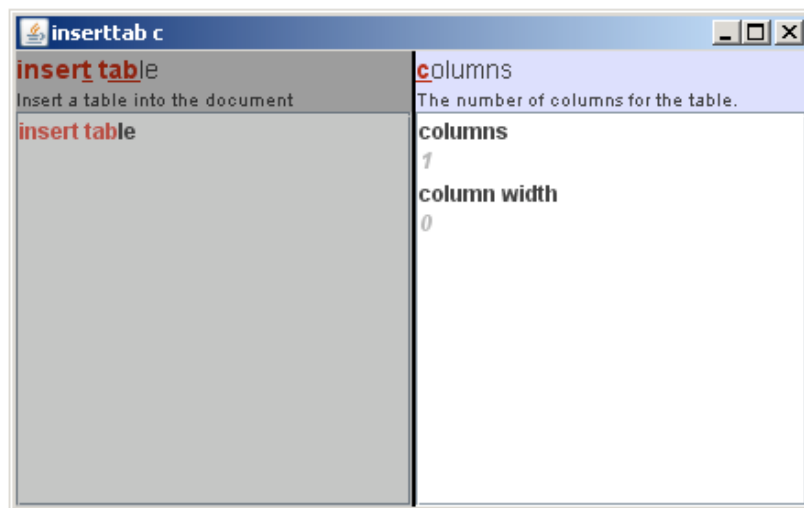


Figure 4-10: GEKA window showing the parameter list for 'insert table' after 'c' has been entered. The best matching parameter is columns. Pressing SPACE will allow a value for that parameter to be selected. The parameter list has been updated to show only parameters that match the input 'c.'

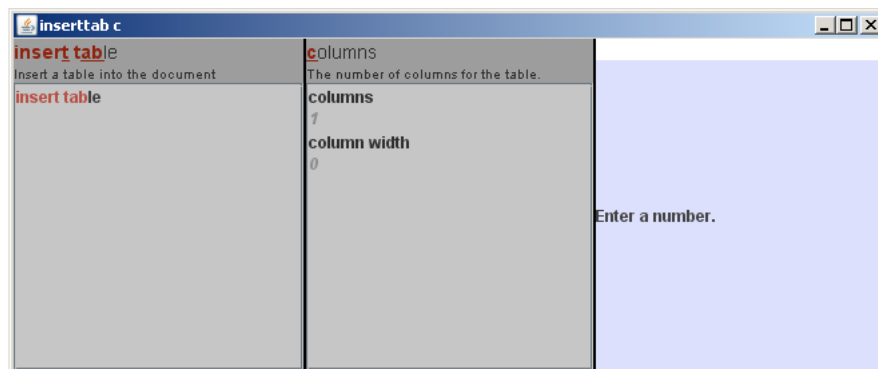
#### 4.2.2.4 Parameter value entry

Parameter value entry is handled in a separate pane. The mechanism used depends on the type of the parameter. We have implemented entry mechanisms for text values (either strings or numbers) and list selection values. The text mechanism is shown in Figure 4-11 for the **columns**

parameter of the **insert table** command. Though not implemented, it would be fairly straightforward to prevent illegal parameter values from being entered by displaying a warning and preventing the parameter value from being accepted. Examples of where this could be useful are preventing letters from being entered into a number field or preventing entry of numbers that are out of the acceptable range.

Figure 4-12 shows the entry mechanism for selecting a value from a list. This mechanism functions just like the command and parameter lists: the user types in characters, and with each one the list is culled and a best match is shown at the top.

For full implementation of a command set, it would be useful to design other specific parameter value mechanisms such as file and colour choosers and simplified input for Boolean values.



**Figure 4-11: GEKA window with the parameter value entry in focus. This third pane in the window allows a value to be entered for “columns.”**



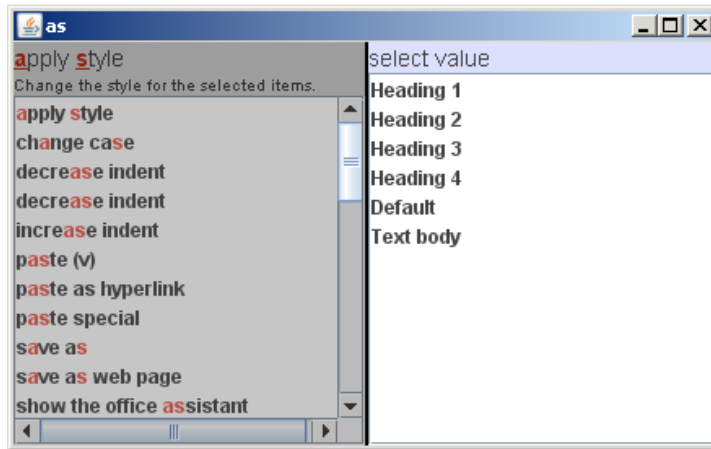
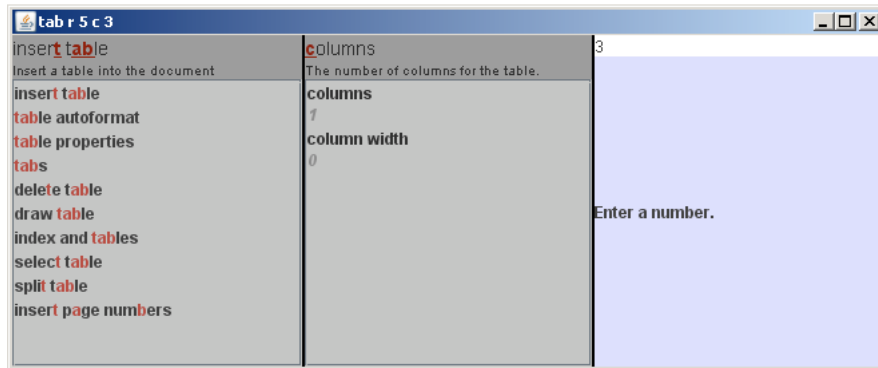


Figure 4-12: GEKA window with the parameter value entry in focus. Parameter values from a list are selected in the same way as command and parameter names. Because the “apply style” command has only one argument, there are only two panels in the window.

#### 4.2.2.5 Text entry area

The full set of characters that have been entered into GEKA is always shown in the title bar of the window. Figure 4-13 shows an example of the text entry area where the text “tab r 5 c 3” has been entered. When a user gains experience with GEKA, learning the short names (or any other sequence of characters that is known to match the desired selection) for commands and parameters, it is possible to ignore the rest of the graphical feedback and just look at the text area as if it were a traditional command line. Very experienced users might go a step beyond this and not look at the GEKA window at all while typing a command and its parameters.



**Figure 4-13: GEKA window demonstrating the text entry area. All characters that have been typed are displayed in the title bar of the GEKA window, in this case “tab r 5 c 3”.**

Any down arrows that have been used to scroll through a list are included in this display, while an up arrow cancels out the previous down arrow and removes it from the display. Because Windows XP does not have full Unicode support, exclamation marks (!) are used to display down arrows rather than the Unicode downward arrow (↓).

### 4.3 Design Discussion

GEKA’s design has three main improvements over existing applications like Quicksilver and Enso.

*Support for multiple parameters in arbitrary order* – GEKA’s use of keyword parameters allows for more flexible commands than other application support. Quicksilver and Enso support only one parameter for each command. Ubiquity has support for multiple parameters, but these are positional parameters, which restrict the flexibility of command input.

*Smarter matching including abbreviations for all commands* – GEKA uses a static matching algorithm whose results are always the same for any given input. This fits better with our goals than does Quicksilver’s adaptive algorithm. Additionally, each command and parameter name in GEKA has a

built-in abbreviation, allowing experienced users to very quickly select the desired item.

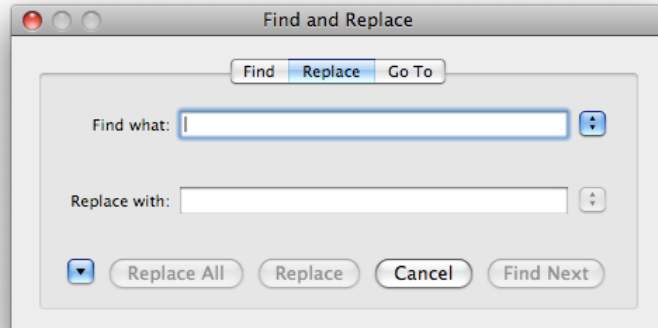
*Clear visual feedback of the input characters* – The title bar in the GEKA window displays all of the characters that have been typed. This allows GEKA to be used more like a traditional command line for advanced users who do not need to look at the graphical feedback. Quicksilver has no display of the input characters. Enso does highlight the input, but it is incorporated into the auto-completed command name and can be difficult to differentiate.

## 4.4 Design limitations

In order to assess how well we have reached our goal of being able to support as many commands as possible, we looked through all of the commands available in the menus and toolbars in Microsoft Word 2003 and identified each of the commands where our current prototype either won't work or will be awkward to use. We grouped these commands into four categories, described below:

*Multi-step dialogs between the user and the computer* – The GEKA mechanism does not work when completing an action that requires multiple phases of input from the user based on output from the computer. A common example of this is wizards. The simplest version of a wizard is just a command with a large number of parameters split across multiple pages. GEKA could certainly handle a command of this type, but often in a wizard, the contents of one page are dependent on the user's input from previous pages. GEKA is not designed to accommodate this kind of multi-step interaction. Another example is find/replace (shown in Figure 4-14), where a dialog box persists on the screen through multiple iterations of executing actions such as **find next** and **replace**. With the current prototype, the GEKA window is closed after each command is executed and there is no

support for find/replace type interaction where multiple commands are executed rapidly with the same set of parameters.



**Figure 4-14: The Find and Replace dialog in Microsoft Word 2008.**

*Inherently visual tasks* – Some actions rely heavily on visual information. These include selecting a document theme or a font colour (Figure 4-15), or inserting a symbol. It would not be a challenge to implement these commands in GEKA, for example, using hex codes for colours resulting in something like “set colour ffee84” or giving names to symbols resulting in “insert symbol n-ary summation.” However, using these commands without any visual preview would likely be quite challenging for users who aren’t experts with the specific command.

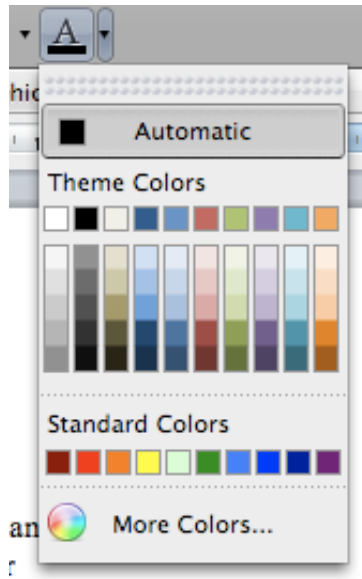


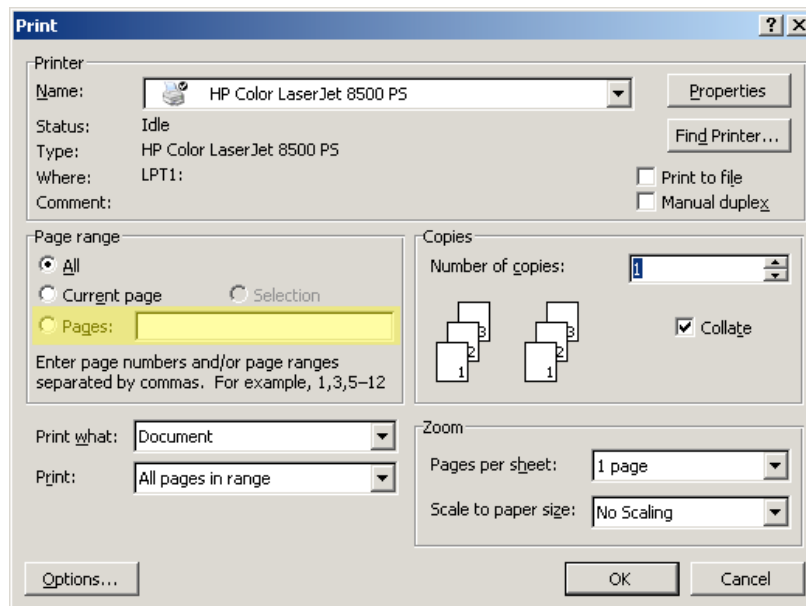
Figure 4-15: The font colour chooser in Microsoft Word 2008.

*Direct manipulation* – Some actions involve directly manipulating objects within a document. One example of this is the format paintbrush (Figure 4-16), where after the command is selected, a portion of the document text must be selected with the mouse to be “painted.” In this case, the command could be easily adapted to work with GEKA by breaking it up into “copy format” and “paste format,” with the selections performed independently of the command (this type of formatting is already supported in Excel). Another example is dragging a table border to change its size. This is similar to the inherently visual tasks described above, where it is possible to create a command such as “set width 400,” but the outcome is difficult to predict without a preview.



Figure 4-16: The “format paintbrush” item in Microsoft Word 2008.

*Nested or multi-part parameters* – Some parameters are not a simple name/value pair. For example, in Word 2003, the **print** command has a **page range** parameter in which one of the values, **pages**, requires further input of the actual page numbers as shown in Figure 4-17. Rather than being a simple name/value pair, this parameter has three parts: name, value, and secondary value. The GEKA command language does not prevent this kind of command from being implemented, because nothing is specified about what a parameter value must be. However, the rigid three-pane layout of our graphical feedback prototype does not allow for this kind of parameter to be fully implemented. A workaround to use this kind of parameter with the current design would be to make the parameter value a freeform text entry, but this would lose all of the benefits of auto-completion and visual feedback.



**Figure 4-17: The print dialog box in Microsoft Word 2003. The highlighted value “pages” for the “page range” parameter requires further input.**

Most of these issues could be addressed with a redesign of the graphical feedback component. One can imagine a graphical component that is less rigid than our prototype allowing more flexibility with the types of parameter

values that can be input, for example showing a colour chooser where values can be easily selected through text input. There could even be a plug-in structure for value entry so that application developers can create parameter value entry mechanisms for specialized parameter types. Furthermore, there could be a signal such as pressing **ALT+ENTER** that executes a command but leaves the window open to facilitate multi-step dialogs. Thus, we feel that the GEKA command language paired with a future iteration of graphical feedback meets our goal of being able to execute most commands.

The prototype design described in this chapter is what was used in our laboratory experiment, described in the following chapter, which explored user's reaction to GEKA and quantitatively compared GEKA with the various WIMP techniques.

## **5 Laboratory Experiment**

We conducted a laboratory experiment to quantitatively examine whether our GEKA prototype was meeting its goals. Specifically, we were interested in testing our design goals of high speed and a low error rate through actual usage data as well as assessing our high-level goal of making computing more pleasant for advanced users by examining their behaviour when given a choice between GEKA and WIMP, and through qualitative questionnaire responses. This chapter describes the experimental design and results.

### **5.1 Experimental environment and tasks**

The experiment used the GEKA design described in the previous chapter. The prototype was initially created as a plug-in for OpenOffice.org Writer. For this experiment, however, we needed greater control over the functionality of the environment than we could get from simple modifications to an existing word processor. We decided to create a replica word processor user interface and to base our replica on Microsoft Word 2003 because it is the best known word processor that uses a standard WIMP interface. The replica contains the full toolbar and menu structure of Word 2003 but a very limited portion of the actual functionality.

The Word replica and the experimental software was coded using C#.NET, while the GEKA prototype was taken from the user interface portion of our OpenOffice.org plug-in, which was written in Java. The Java and C#.NET programs communicate with each other through standard input/output redirection.



### 5.1.1 Commands

The design for the experiment was based on comparing specific WIMP methods to their GEKA alternatives. The WIMP methods used in the experiment were slightly different than those used in the formative study. Toolbars were broken down into two categories, toolbar buttons and toolbar drop-downs, because these are in fact quite different methods, with the first executing simple zero-parameter commands and the latter involving the selection of one parameter. Our formative study showed that dialog boxes often frustrate users, so dialog boxes were added to the WIMP methods in this study and compared to multi-parameter GEKA commands. Finally, context menus were removed because the formative study showed that they are rarely used. The study thus examined five types of WIMP commands: keyboard shortcuts, toolbar buttons, menu items from the menu bar, toolbar drop-downs, and dialog boxes. Throughout this chapter, we will refer to WIMP methods and their GEKA equivalents. Keyboard shortcuts, toolbar buttons, and menu bar items were compared to using GEKA with zero-parameter commands, toolbar drop-downs were compared to GEKA with one-parameter commands, and dialog boxes were compared to GEKA with multiple-parameter commands. This is shown in Table 5-1.

**Table 5-1: The WIMP methods and their GEKA equivalents used in this experiment.**

<b>WIMP Method</b>	<b>GEKA Equivalent</b>
Keyboard shortcut	Zero-parameter commands
Toolbar button	Zero-parameter commands
Toolbar drop-down	One-parameter commands
Menu bar item	Zero-parameter commands
Dialog box	Multiple-parameter commands

To examine these types of commands, we needed to select a set of commands for each method that could be executed in the experiment using

both the WIMP method and the corresponding GEKA method. The commands that we chose are representative of their WIMP method and should be familiar to advanced Microsoft Word users. We used the following 15 commands for the experiment.

- Keyboard shortcuts – underline, italic, copy
- Toolbar buttons – bold, center alignment, toggle bullets
- Toolbar drop-downs – font size, apply style, line spacing
- Menu bar commands – paste, undo, save
- Dialog boxes – print, insert table, insert page

Many of these commands can be executed in Microsoft Word through several WIMP methods. Because we needed an equal number of command executions with each method, every command was restricted to only being able to be executed through the method for which it is listed above. Some of the command-to-method mappings are not what we would expect an advanced user to typically choose. For example, executing paste through the menu bar is very rare for advanced users. These unexpected mappings were chosen primarily due to low numbers of commands in some categories, i.e. there are few menu bar commands that are familiar to most users and have no parameters.

The commands listed above were fully implemented in our experimental software for the listed WIMP method and the corresponding GEKA method. To create a realistic environment, the full toolbar and menu bar contents of Microsoft Word 2003 were replicated in our user interface. However, none of the entries other than those listed above had any functionality. Similarly, all toolbar items and top-level menu bar items were added to the GEKA command list, but only those listed above had functionality. Selecting a command other than these had no effect.

## 5.2 Task

Because using GEKA involves typing a portion of the command name, displaying these names to the participants during the trials could bias the results. Thus, we had to create an alternative method to prompt participants on which command to execute. We created a series of images that correspond to each command. To prevent bias for the WIMP condition, the images needed to avoid resembling the icons used in Word. The text formatting images were very straightforward. They simply showed the effect of the command on the text, as illustrated in Figure 5-1 for the command **bold**. Commands with no clear effect on document contents required more abstract images, as illustrated in Figure 5-2 for the command **copy**. When the precise effect of a command was ambiguous, such as which exact size the font was being changed to, that information was included in the image, as seen in Figure 5-3 for the command **font size 24**. Finally, commands with multiple parameters required a description of each parameter as illustrated in Figure 5-4 for the **print** command. The full list of images is in Appendix B.

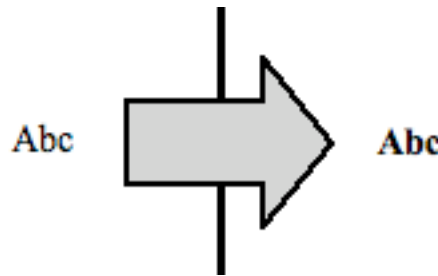


Figure 5-1: Command image for 'bold'.



Figure 5-2: Command image for 'copy'.

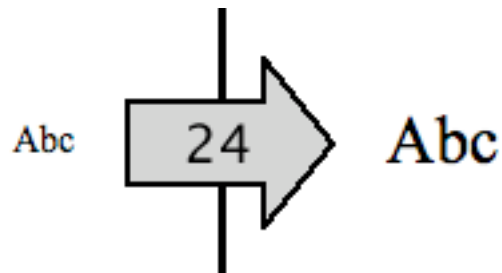


Figure 5-3: Command image for 'font size 24'.



Figure 5-4: Command image for 'print copies 2 pages selection'.

Figure 5-5 shows a screenshot of the experimental application. On the right side of the screen is the replica of the Microsoft Word user interface, showing the toolbar, menu bar, text area, and the dialog box for the **insert page numbers** command. On the left of the screen is an instructions window that tells the participant which action to perform. Each set of instructions is called a task, and consists of a text selection followed by four

command executions. The text to be selected is indicated in the first image of the window, and the four commands appear below it. The actions must be performed in the correct order. After an action is completed, the blue highlight moves down the list indicating which action is to be performed next. Beside each command in the instruction window is a piece of text indicating which method must be used to execute the command. If an incorrect action was performed, nothing happened. The participant was required to continue trying until the correct action was performed. Each incorrect attempt was considered an error.

We created five document editing tasks using the commands described above. The tasks are sets of commands that could reasonably be executed sequentially in actual word processor usage. Each task contains a mixture of the WIMP methods. The tasks are as follows.

- Task A – bold, italic, paste, insert table
- Task B – font size, underline, save, print
- Task C – insert page numbers, undo, apply style, center alignment
- Task D – font size, toggle bullets, line spacing, italic
- Task E – insert page numbers, undo, bold, copy

The grouping of commands into tasks is not relevant to our hypotheses or analysis. It was done simply to present commands to the participants in reasonable chunks. The only important factor in the task design was that each WIMP method appears an equal number of times across the five tasks. With 15 commands and 20 slots in the tasks, it was necessary that five of the commands appear twice. One command from each WIMP method is used twice in the tasks, resulting in exactly 4 executions of each WIMP method in the tasks (bold, insert page numbers, italic, and undo).

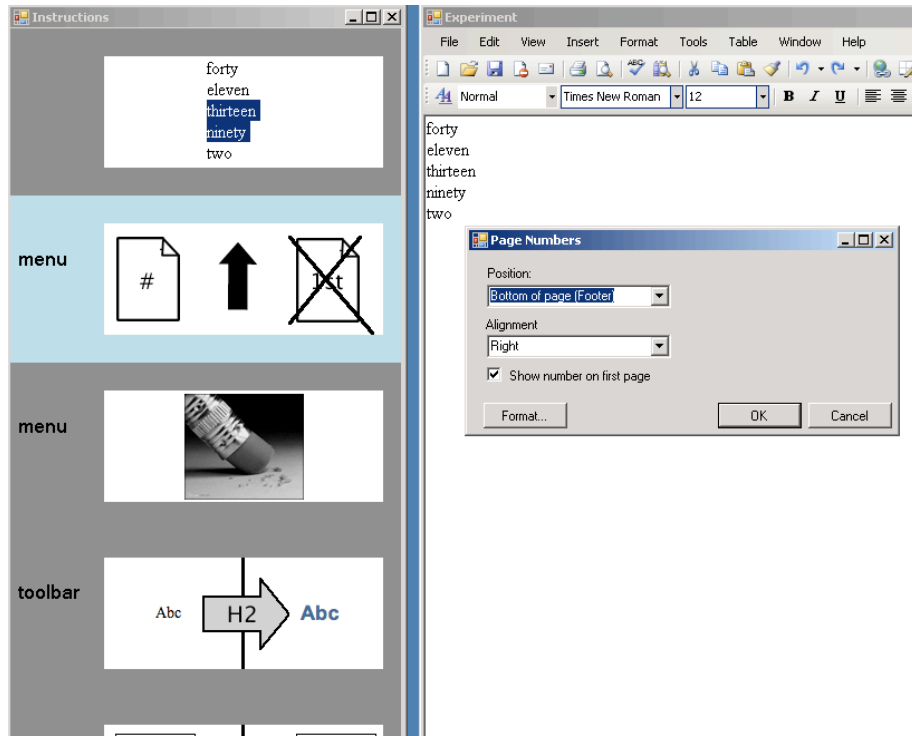


Figure 5-5: Experimental application

### 5.3 Participants

As with our formative study, we were interested primarily in studying advanced computer users for this laboratory experiment. Our 12 participants (3 female) were all graduate students, 3 in computer science, 6 in electrical and computer engineering, and 3 in mechanical engineering. In addition to those 12 participants, we had 8 pilot participants whose data was not used in the analysis. There were two regular participants who were not able to finish all of the tasks on time. Their data was not included in the analysis. Two replacement participants were recruited whose data was used. All participants had significant experience with Microsoft Word 2003. Participants were compensated \$30 for their time and the top 1/3 in terms of performance were given an additional \$10.

## 5.4 Procedure

Each participant completed a single three-hour session in which four distinct phases were completed.

*Phase 1: Introduction* – After a brief description of the session, participants were given a list containing all 15 of the commands used in the experiment and asked to indicate which of the 5 WIMP methods that they knew how to use for each command. They were then given a demonstration of how to use GEKA. Next, they were given sheets of paper containing all of the images used to represent commands in the study alongside the command name and any parameter names and values that the images represented. Participants were asked to review the sheets until they were comfortable with each of the image-to-command pairings.

*Phase 2: Performance testing* – This phase was the bulk of the experiment. Participants completed a number of repetitions of the five tasks described above in two separate conditions: one using only WIMP methods and one using only GEKA. The presentation order for the two conditions was counterbalanced.

Each condition began with a simple practice block that was not included in the analysis. These practice blocks were designed to ensure that the participant knew how to execute each command that was required. Practice blocks contained each of the 15 commands one time. During the practice blocks, participants were able to look at the sheets containing the image-to-command-name mappings and to ask questions of the researcher. Neither of these aids was allowed in subsequent blocks. After the first practice block, participants were given an overview of how the rest of the session would proceed and they were reminded of the monetary prize for top performers.

Each condition consisted of three blocks. Each block contained 10 back-to-back repetitions of each of the five tasks. The order of the five tasks was randomized for each participant and remained consistent across all blocks for each participant. Within a block, after the 10 repetitions of each task were completed, participants took a break for at least 30 seconds. After each block, participants took a break for at least 90 seconds. There was no upper time limit on the breaks. Participants were provided with magazines to peruse during the breaks.

*Phase 3: Method choice* – After the WIMP and GEKA conditions, participants were given one final block of tasks in which they could choose any WIMP method or GEKA to execute each command. Because participants had grown used to executing each command with only one specific method, they were first given an exercise to remind them of all the methods they knew for each command. The researcher read through the list of commands and method types that the participant filled out in the introduction, and the participant executed each command once with each WIMP method and once with GEKA. In the method choice block, each task was repeated only three times, and participants could choose any method to execute each command each time.

*Phase 4: Qualitative feedback* – In the final phase of each session, participants completed a questionnaire in which they rated various aspects of their interaction experience with GEKA and WIMP throughout the experiment.

## **5.5 Piloting**

The above description of the experimental procedure includes improvements to the design whose need became apparent through piloting. Initially, the order of tasks within each block was completely randomized for each participant, with no forced back-to-back repetition. In the early piloting,



it became apparent that participants were spending too much time deciphering the instructions for each task, especially in the WIMP condition, where two pieces of information were required for each command: the command itself and the necessary method. To remedy this, we switched to the back-to-back repetitions. First, we tried five blocks with five repetitions of each task in each block. With this design, there was no plateau within the five repetitions of each task, so it was clear that even by the fifth repetition, participants were still spending time deciphering the instructions. We then moved to the final design using three blocks each containing ten back-to-back repetitions of each task.

## 5.6 Dependent measures

Time was recorded for each command from the moment that the previous command was completed until the current command was correctly completed. This includes an implicit error penalty because the timer was not reset after an error occurred.

Because our hypotheses dealt with interaction methods rather than specific commands, the time measure was collapsed into a single time for each interaction method. Each interaction method was used four times in the set of five tasks. Each of the tasks was repeated ten times per block. The final time measure for each repetition of each interaction method is the mean of its four executions in that particular repetition. For example, the menu bar time for repetition 1 is the average of the **paste** command from repetition 1 of task A, the **save** command from repetition 1 of task B, the **undo** command from repetition 1 of task C, and the **undo** command from repetition 1 of task E. Similarly, a menu bar time for repetition 2 was calculated using the second repetition of the same commands.

The number of errors was also recorded for each command. An error consisted of trying to execute an incorrect command, or the correct command

with incorrect parameters. For example, clicking on **redo** instead of **undo** in the menu bar, printing 4 copies instead of 3 copies, or pressing **CTRL+V** instead of **CTRL+C** were errors. Incorrect actions that did not result in a command execution were not considered errors: opening a dialog box and then clicking cancel, clicking the wrong menu heading but not actually selecting anything in the menu, or pressing an invalid key combination such as **CTRL+SHIFT**.

Once again, our hypotheses related to errors dealt with interaction methods rather than specific commands. Thus, an error value for each interaction method was computed similarly to the time value described above. The one difference is that the error value for each repetition is the sum of the errors for each of the four command executions instead of the mean.

Method choice was measured for each command as the method used in the final repetition of each task during the method choice phase. The final repetition was used because participants often changed methods during the repetitions. The results were very similar when analyzed using all repetitions.

Finally, the questionnaire had participants rate each of the methods on a scale resembling the NASA TLX on 12 dimensions: ease of learning, ease of remembering, physical demand, mental demand, visual demand, effort, tediousness, frustration, distraction, speed, error rate, and overall opinion. The actual wording and scale used in the questionnaire is in Appendix C. The five WIMP methods were included as well as three cases for GEKA: zero-, one-, and multi-parameter commands.

## 5.7 Motivation

To motivate participants to execute commands as quickly and accurately as possible, an additional \$10 prize was awarded to the top-third-highest performers in the experiment. The one-third ratio was chosen so that participants would know they had a reasonable chance of winning.

## 5.8 Hypotheses

We had the following hypotheses, which are consistent with our goals in designing GEKA.

- **H1:** Command selection in GEKA will be faster than and preferred to menu selection.
- **H2:** Command selection in GEKA will be slower than and will not be preferred to keyboard shortcuts.
- **H3:** For commands with multiple parameters, GEKA will be faster than and preferred to dialog boxes.
- **H4:** For commands with one parameter, users will prefer GEKA to toolbar drop-downs.
- **H5:** GEKA will be no more error-prone than WIMP.

Here, preference refers to both explicit method choice and the qualitative questionnaire ratings.

## 5.9 Design

Data was analyzed using the collapsed time and error measures described in section 5.6. The following mixed factor design was used: 2 (conditions) x 3 (blocks) x 5 (interaction methods) x 10 (repetitions) x 2 (presentation orders). Presentation order was a between-participants factor, while all others were within-participants factors.

Pairwise comparisons used Bonferroni corrections. When sphericity was an issue with our data, Greenhouse-Geisser corrections were used, which can be identified by non-integer *df*.

## 5.10 Results

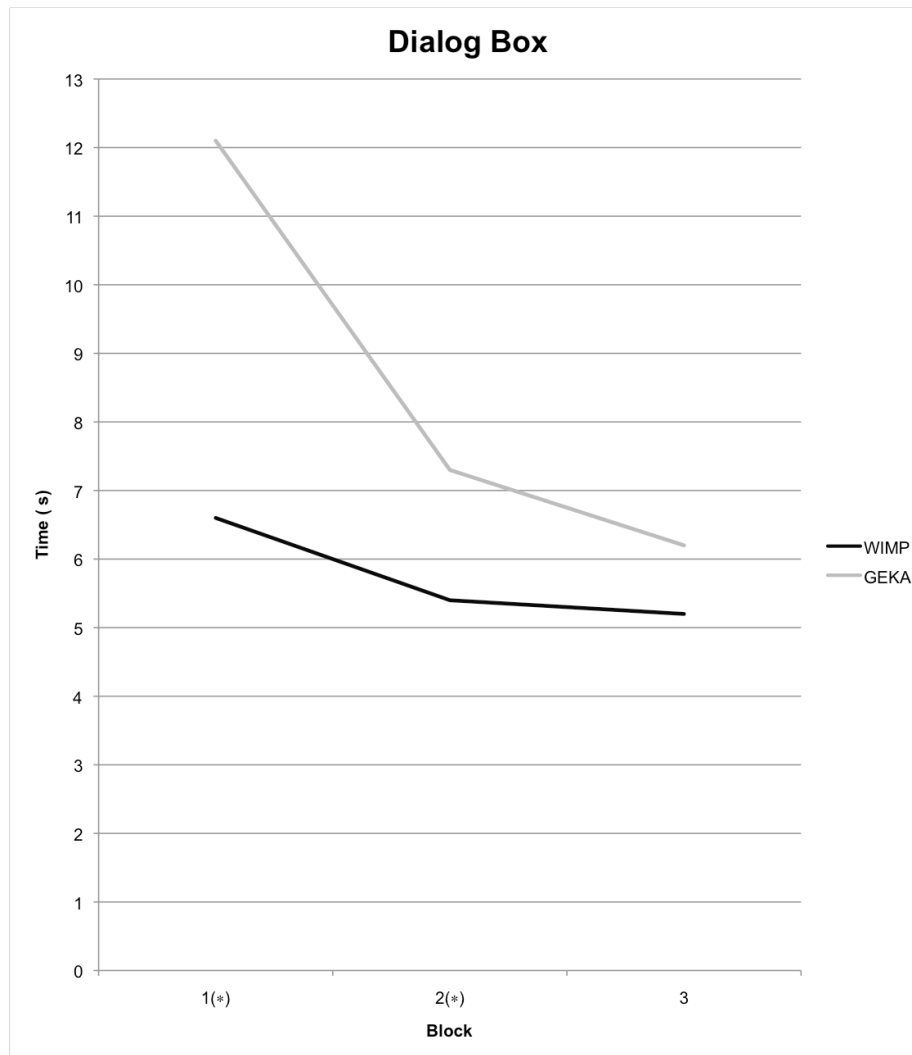
Initial analysis showed no significant main or interaction effects of presentation order, so presentation order was dropped as a factor to simplify further analysis.

### 5.10.1 Time

There was a significant main effect of condition ( $F(1, 11) = 13.320, p = .004, \eta^2 = .548$ ), with WIMP (mean 2.719s) being overall faster than GEKA (mean 3.645s). There was also a significant main effect of block ( $F(1.12, 12.35) = 115.610, p < .001, \eta^2 = .913$ ), showing that learning was occurring across blocks. An interaction effect between block and condition ( $F(2, 22) = 53.670, p < .001, \eta^2 = .830$ ), shows that GEKA improved more rapidly across blocks than did WIMP. This may explain a major part of WIMP's overall time advantage.

The most interesting time effect was a significant three-way interaction between condition, block, and interaction method ( $F(2.65, 29.13) = 27.604, p < .001, \eta^2 = .715$ ). Figure 5-6 through Figure 5-10 illustrate this interaction with a graph for each interaction method. Table 5-2 shows the full means and significance for this interaction.

The pairwise comparisons show that in block 1 each WIMP method was faster than its GEKA equivalent (all  $p < .05$ ), except for menu, which showed no significant difference ( $p = .817$ ). By block 3, GEKA performance had improved dramatically in comparison to WIMP. GEKA was significantly faster than menus ( $p < .05$ ), while there was no significant difference for toolbar drop-downs,  $p = .503$ , or dialog boxes,  $p = .102$ . WIMP remained faster for shortcuts and toolbar buttons (both  $p < .05$ ).



**Figure 5-6: Dialog box times for WIMP and GEKA in each block. Blocks are labeled with a \* when the time differences are statistically significant ( $p < .05$ ) ( $N=12$ ).**

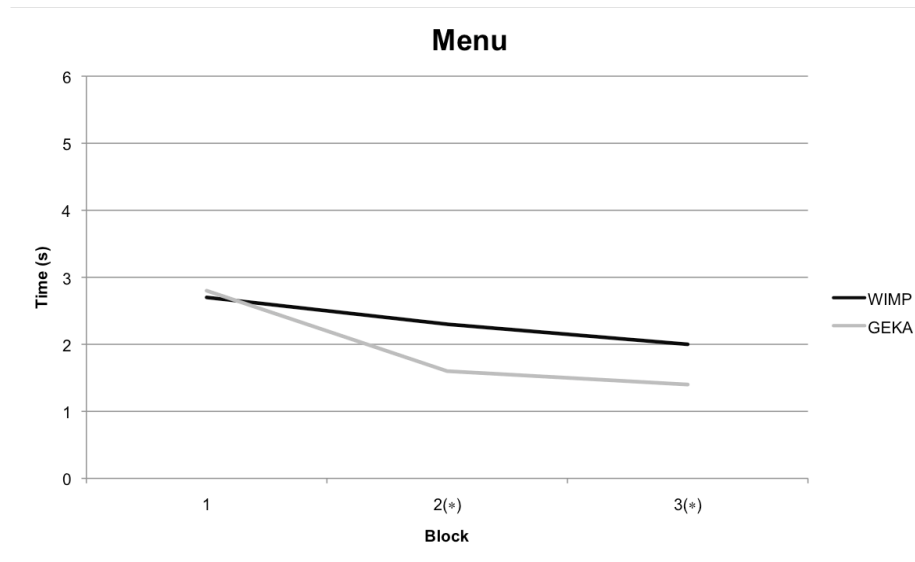


Figure 5-7: Menu bar times for WIMP and GEKA in each block. Blocks are labeled with a \* when the time differences are statistically significant ( $p < .05$ ) ( $N=12$ ).

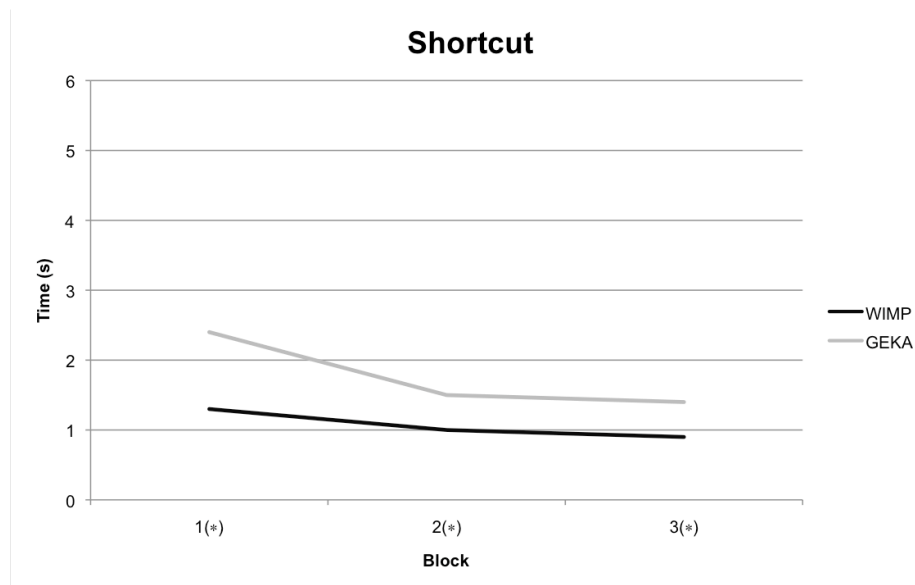
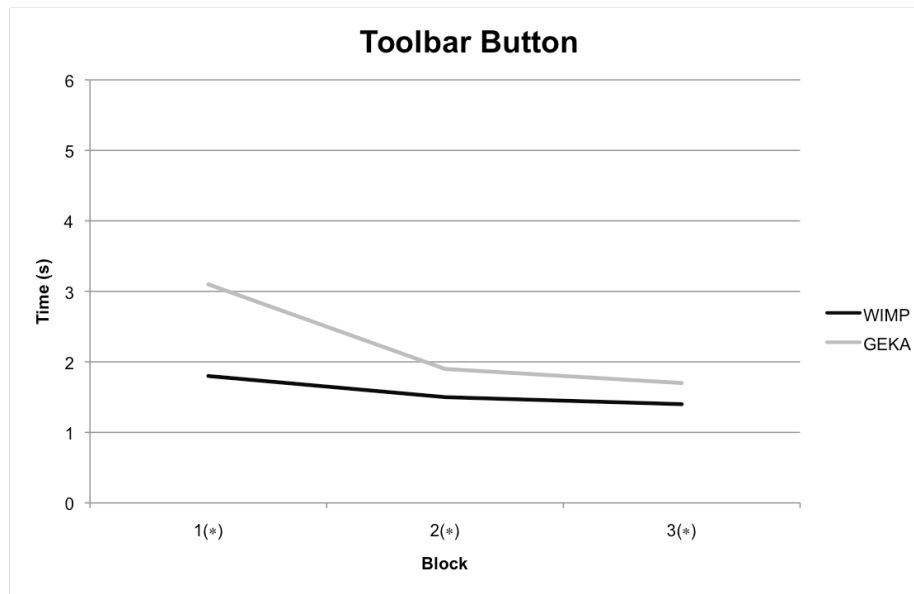
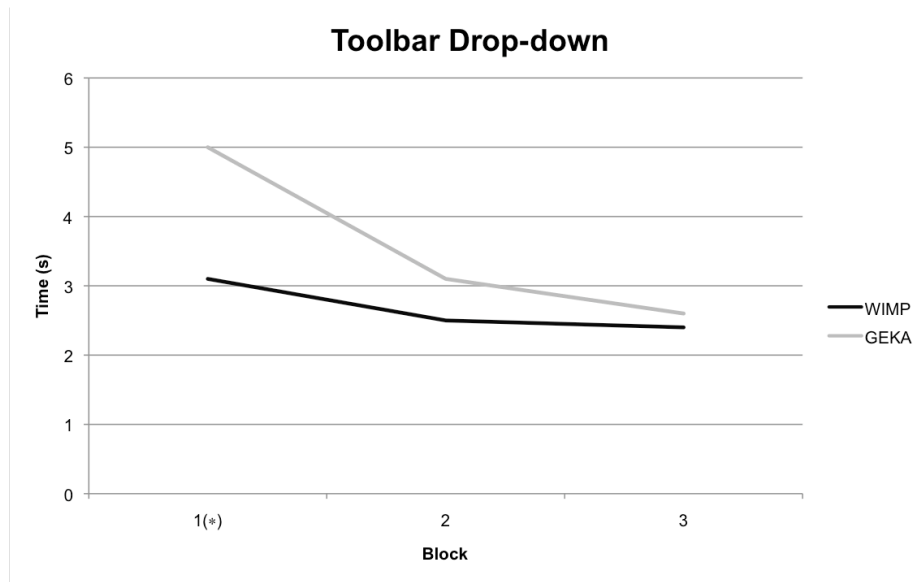


Figure 5-8: Keyboard shortcut times for WIMP and GEKA in each block. Blocks are labeled with a \* when the time differences are statistically significant ( $p < .05$ ) ( $N=12$ ).



**Figure 5-9: Toolbar button times for WIMP and GEKA in each block. Blocks are labeled with a \* when the time differences are statistically significant ( $p < .05$ ) ( $N=12$ ).**



**Figure 5-10: Toolbar drop-down times for WIMP and GEKA in each block. Blocks are labeled with a \* when the time differences are statistically significant ( $p < .05$ ) ( $N=12$ ).**

**Table 5-2: Breakdown for the interaction between condition, block, and interaction method. Times are in seconds (N=12).**

<b>Block</b>	<b>Method</b>	<b>WIMP Time (s)</b>	<b>GEKA Time (s)</b>	<b>Significance</b>
1	Shortcuts	1.305	2.452	.000
	Toolbar buttons	1.877	3.158	.000
	Toolbar drop-downs	3.139	5.060	.002
	Menus	2.741	2.800	.817
	Dialog boxes	6.663	12.112	.000
2	Shortcuts	1.022	1.527	.001
	Toolbar buttons	1.497	1.976	.004
	Toolbar drop-downs	2.568	3.100	.116
	Menus	2.312	1.649	.000
	Dialog boxes	5.496	7.378	.014
3	Shortcuts	.930	1.411	.000
	Toolbar buttons	1.435	1.736	.047
	Toolbar drop-downs	2.478	2.651	.503
	Menus	2.069	1.430	.000
	Dialog boxes	5.248	6.237	.102

To determine whether performance was still improving, we examined the difference between blocks 2 and 3 in the above condition-method-block interaction. In the WIMP condition, there was a significant difference only for toolbar buttons ( $p = .042$ ) and a borderline difference for menus ( $p = .057$ ). This indicates that the other three interaction methods had plateaued: participants were no longer improving. For the GEKA equivalents, there were significant differences for toolbar buttons, drop-downs, menus, and dialog



boxes (all  $p < .05$ ), showing that performance was still improving for all but one method (shortcuts).

Additionally, but not surprisingly, there were significant main effects of repetition ( $F(1.79, 19.68) = 65.505, p < .001, \eta^2 = .856$ ) and interaction method ( $F(1.14, 12.57) = 216.641, p < .001, \eta^2 = .953$ ), and an interaction between condition and interaction method ( $F(1.27, 13.96) = 19.670, p < .001, \eta^2 = .641$ ).

### 5.10.2 Errors

The analysis for errors showed no significant difference for condition, with a total of 254 errors in WIMP and 256 in GEKA ( $F(1, 11) = .004, p = .948, \eta^2 < .001$ ). There was a borderline significant main effect of block ( $F(2,22) = 3.322, p = .055, \eta^2 = .232$ ), but no interaction between block and condition. There was a significant interaction between condition and interaction method ( $F(4,44) = 3.260, p = .020, \eta^2 = .229$ ), with pairwise comparisons showing two borderline significant differences: GEKA had more errors than dialog boxes ( $p = .071$ ) and fewer errors than toolbar drop-downs ( $p = .085$ ). Table 5-3 shows the full breakdown for this interaction.

**Table 5-3: Total errors. Each method was used 1440 times per condition ( $N = 12$ ).**

	WIMP	GEKA	Sig.
Shortcut	30	33	0.845
Toolbar button	15	23	0.136
Toolbar drop-down	91	46	0.085
Menu bar	43	31	0.394
Dialog box	75	123	0.071
Total	254	256	0.948

### 5.10.3 Method choice

Figure 5-11 shows how often each interaction method was chosen for each of the 15 commands in the method choice phase of the experiment.

Participants overwhelmingly chose to use the two keyboard-based methods: GEKA and keyboard shortcuts. For commands that have parameters and thus cannot be completed fully with keyboard shortcuts, GEKA was used nearly all of the time. For commands that don't have parameters, shortcuts were chosen the majority of the time, except for two commands: **bullets** and **center**. The shortcuts for these commands were generally unknown to our participants (one knew **center** and none knew **bullets**). Rather than resorting to a mouse-based method, participants chose GEKA for all instances of **bullets** and the majority of **center**.

In short, keyboard shortcuts were used where they were known, GEKA was chosen where shortcuts weren't known.

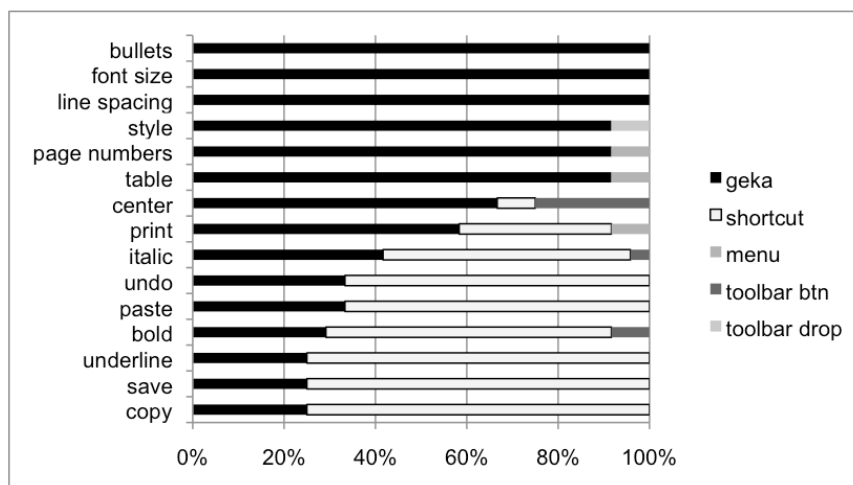


Figure 5-11: Percentage of command executions using each method in the method choice phase of the experiment ( $N=12$ ).

#### 5.10.4 Qualitative findings

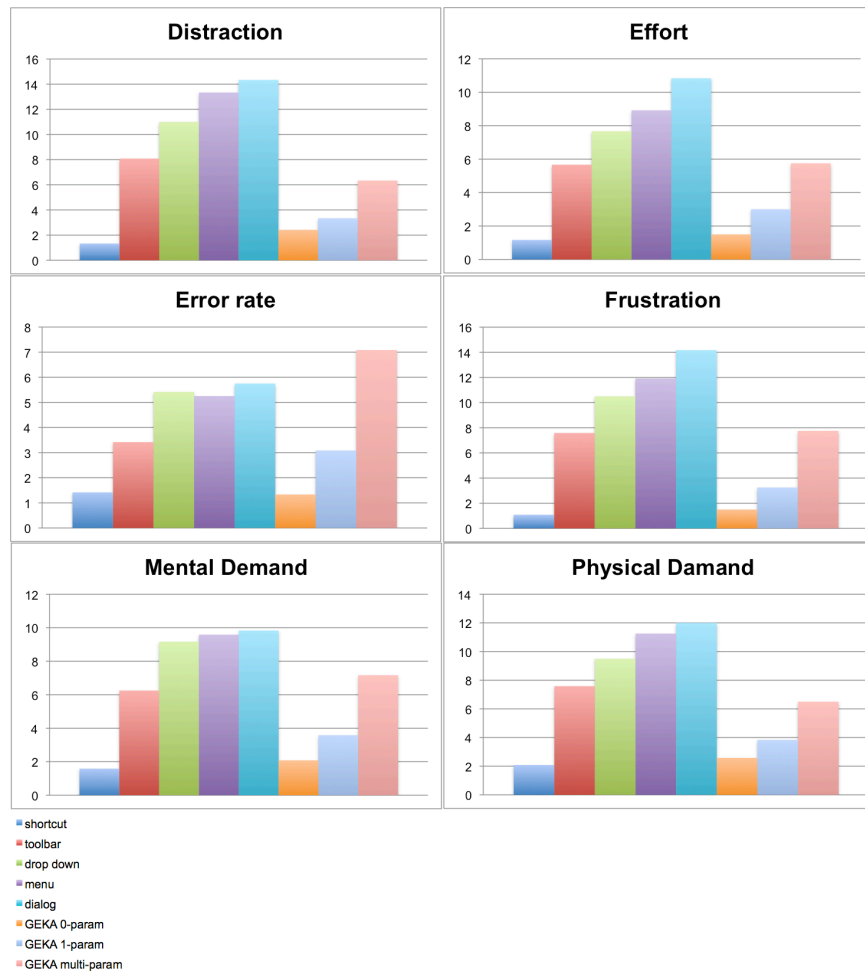
Figure 5-12 and Figure 5-13 show the full questionnaire results. For each dimension, participants rated each interaction method individually. A lower score is better.

The responses on each dimension appear to be very similar. This was confirmed by a reliability analysis showing high consistency (Cronbach's  $\alpha = .966$ ). Because of this similarity, we collapsed the data into a single measure

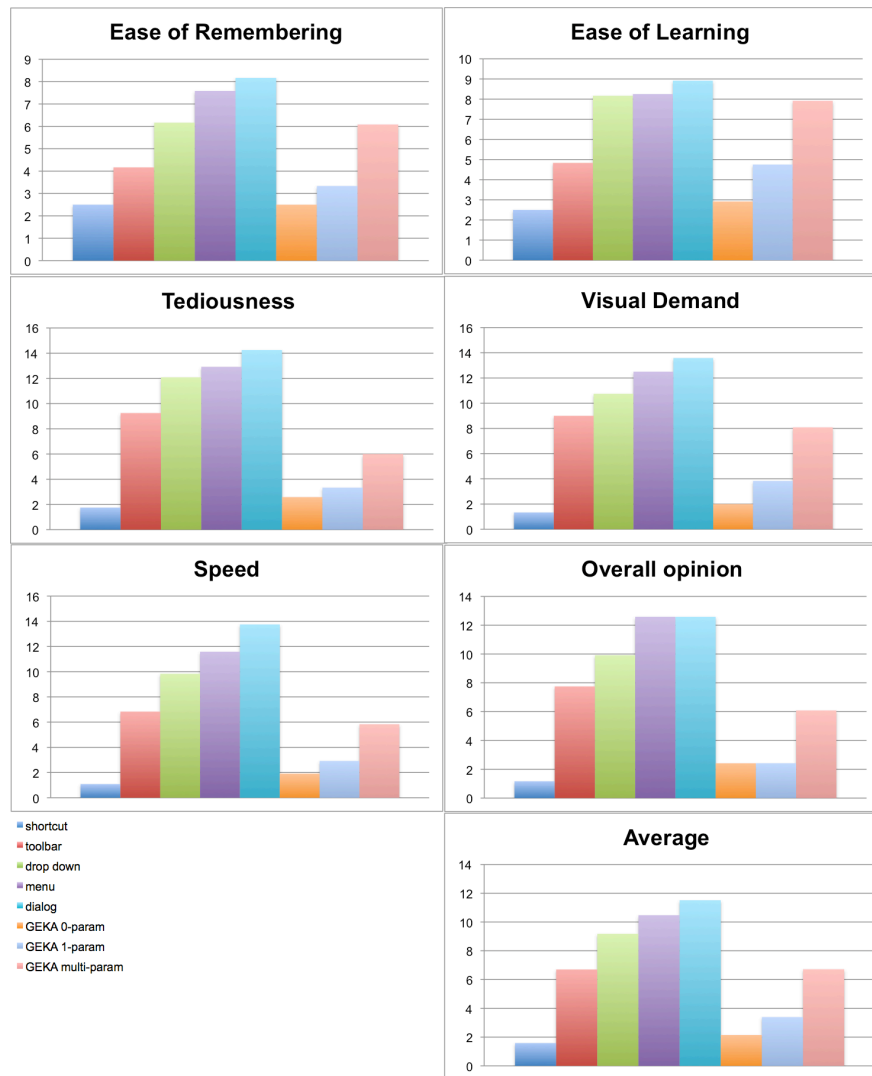
by averaging the responses on each dimension, shown in the last chart of Figure 5-13. A Friedman test on these averaged ratings showed a significant main effect of interaction method ( $\chi^2(7)=63.328$ ,  $p < .001$ ) and Wilcoxon Signed Rank Tests showed significant differences ( $p < .05$ ) between each GEKA method and its corresponding WIMP method: between GEKA zero-parameter and both toolbar and menu, between GEKA one-parameter and dropdown, and between GEKA multi-parameter and dialog box. The one exception is that there was no significant difference between GEKA zero-parameter commands and keyboard shortcuts ( $p = .155$ ).

Thus, GEKA was preferred overall to each WIMP method except keyboard shortcuts, to which it was comparable.

There are two interesting points to be drawn from the individual charts. First, the error rate responses, shown in Figure 5-12, are the only place where GEKA multi-parameter commands were rated worse than dialog boxes. This is in fact consistent with the performance data on error rate discussed above. The speed responses, shown in Figure 5-13 – consistent with the rest of the responses – show GEKA to be better rated than all WIMP methods except keyboard shortcuts. This is interesting because it conflicts with the actual performance data on speed, which had mixed results with GEKA even being slower in some cases. It appears that participants felt GEKA was faster than it actually was.



**Figure 5-12: Ratings for WIMP and GEKA methods from the qualitative feedback phase of the experiment, part 1 of 2. Lower scores are better (N=12).**



**Figure 5-13: Ratings for WIMP and GEKA methods from the qualitative feedback phase of the experiment, part 2 of 2. Lower scores are better (N=12).**

### 5.10.5 Summary of results

We summarize our results according to our hypotheses:

- **H1:** Command selection in GEKA will be faster than and preferred to menu selection. **Supported.** *GEKA zero-parameter commands were significantly faster than WIMP menu selections. In the method choice phase and the questionnaire, GEKA was overwhelmingly preferred.*
- **H2:** Command selection in GEKA will be slower than and will not be preferred to keyboard shortcuts. **Supported.** *Keyboard shortcuts in the WIMP condition were significantly faster than GEKA zero-parameter commands. Keyboard shortcuts were generally used when they were known in the method choice phase and were rated more highly than GEKA on the questionnaire.*
- **H3:** For commands with multiple parameters, GEKA will be faster than and preferred to dialog boxes. **Partially supported.** *There was no difference in speed, but GEKA was preferred in the questionnaire and method choice phase.*
- **H4:** For commands with one parameter, users will prefer GEKA to toolbar drop-downs. **Supported.** *GEKA one-parameter commands were consistently chosen over toolbar drop-downs in the method choice phase were rated better in the questionnaire.*
- **H5:** GEKA will be no more error-prone than WIMP. **Supported.** *There was no significant difference in errors between GEKA and WIMP.*

## 5.11 Discussion

We conclude that GEKA's speed is very competitive with WIMP. GEKA is faster than menus and no different in speed than the other mouse-based WIMP techniques except for toolbar buttons. This is after only a short

exposure to GEKA. In most cases, participants were shown to be still improving. It is possible that these results would be even more encouraging for GEKA after more practice.

GEKA had comparable error rates to WIMP. Analyzing the errors by interaction method, there are interesting differences that need to be examined further, but overall, there is almost no difference in error rates. This is promising, considering that our participants have been using GUIs for a number of years but had no prior GEKA experience.

GEKA was overwhelmingly preferred to mouse-based WIMP methods. Both the method choice and qualitative feedback phases of the experiment showed very strong preference for GEKA over all WIMP methods except keyboard shortcuts.

In the method choice phase of the experiment, GEKA was consistently used instead of toolbar buttons, toolbar drop-downs, dialog boxes, and menus, even though it was only faster than menus. This is consistent with responses on speed in the questionnaire. It is possible that those responses were biased by participants' desire to please the researchers, but this is much less likely in the method choice phase because participants were aware that there was a monetary reward contingent on their performance and were thus motivated to select what they truly felt was the fastest method.

We have as yet no firm basis to know why GEKA feels faster than it truly is. We speculate that our participants found GEKA more pleasant to use than WIMP and therefore time spent using GEKA might seem to pass more quickly, but further work is needed to tease this apart.

This finding on perception of speed causes us to reflect back on our original goal that GEKA should be faster in order to be attractive to advanced users, a goal consistent with others working in this design space, such as Raskin (2000). Achieving pleasurable use, with no notable degradation in speed, is quite possibly just as important, or perhaps more important than a

speed improvement alone.



## 6 Conclusions and Future Work

This thesis presents research on the design and evaluation of Graphically Enhanced Keyboard Accelerators (GEKA). The goal of this work is to make interactive desktop computing more pleasant and productive for experienced computer users by giving them the option to execute commands and specify parameters quickly and easily using the keyboard. Our work builds on design ideas from a number of similar applications, introducing improvements that support our design goals of completeness, low visual demand, ease of learning and remembering, low error rate, and high speed. Additionally, we provide the first formal evaluation of this type of technique.

Our formative study explored how our target users currently work with WIMP interfaces. We confirmed our suspicions that advanced computer users prefer to use the keyboard to execute commands but are often not able to do so either because there is no keyboard method available or because they have not been able to learn or remember the available method. We identified dialog boxes as a particularly problematic interface component for which nearly all users resort to the mouse despite their stated desire to use a keyboard. The formative study confirmed our notion of the GUI gap: the unexplored design space between the easy but slow mouse-based methods and the fast but incomplete and difficult keyboard-based methods. This suggests that there is in fact a desire among advanced users for a GEKA-like interaction method and it motivates our research in this area. Finally, the formative study provided initial feedback on an early GEKA prototype, helping us to refine the final design that is presented in this thesis.

GEKA's design blends several elements from WIMP and CLIs. It uses a prefix command language with keyword parameters and an auto-completion mechanism based on incremental search to allow commands to be entered

quickly and precisely. Additionally, it makes use of graphical feedback to show what values have been selected and what options are available, shifting the focus of interaction to recognition rather than recall. This design builds on applications such as Quicksilver and Enso, which also combine command languages with graphical feedback and incremental search. GEKA has three key improvements over existing systems: support for multiple parameters in arbitrary order, smarter matching – including abbreviations for all commands and tolerance for “sloppy typing,” and clear visual feedback of the input characters to facilitate learning and re-use.

Our laboratory experiment was the first evaluation of any of the recent applications supporting command language interaction in graphical environments. Participants in the experiment were all from our target user group of advanced computer users. The primary goal of the experiment was to compare zero-, one-, and multiple-parameter commands in GEKA to the equivalent WIMP interaction methods in terms of speed and error rate. We found overall error rates to be nearly identical. Examining error rates by interaction method showed two borderline significant differences: GEKA multiple-parameter commands had roughly twice as many errors as dialog boxes, but GEKA one-parameter commands had about half the errors of toolbar drop-downs. In terms of speed, GEKA was not designed to compete with keyboard shortcuts and thus it was, not surprisingly, slower. GEKA was very competitive with the speeds of the mouse-based WIMP methods, being faster than menus and showing no statistically significant difference from dialog boxes and toolbar drop-downs. Toolbar buttons, however, were faster than their GEKA equivalent.

The experiment also explored users’ preferences between WIMP and GEKA through a series of tasks where users could choose which interaction method to use for each command, and through a questionnaire. When given a choice between WIMP and GEKA, participants overwhelmingly used keyboard shortcuts when they knew them and used GEKA when they didn’t

know the shortcuts. In the questionnaire, each type of GEKA command was rated better than its WIMP equivalent except for zero-parameter GEKA commands relative to keyboard shortcuts. These results suggest that our target user population may have a strong preference for GEKA over the mouse-based WIMP methods.

## 6.1 Discussion and limitations

A surprising result from the study was that the questionnaire and method choice data both indicate that participants feel GEKA is faster than all of the mouse-based WIMP methods. This contradicts the quantitative data, which shows no significant time difference in most cases and GEKA actually performing worse than toolbar buttons. Further study is required to explore this disconnect.

While GEKA speeds were competitive with mouse-based WIMP methods, the results were not as strong as we had expected. Part of this may be due to the fact that GEKA performance was shown to be improving more than WIMP performance at the end of the experiment. Our experiment sessions were three hours long and could not have been reasonably extended to add more repetitions without running a risk of fatigue effects. It is possible that extended exposure to GEKA, through a longitudinal multi-session study, would show GEKA performance continuing to improve compared to WIMP. Furthermore, given the strong preference for GEKA, it is clear that GEKA has advantages over WIMP beyond raw speed. Further research will help us understand these advantages.

The current method of using **CTRL+ENTER** to open GEKA and **ENTER** to execute the selected command could be improved to speed up GEKA interaction. The easiest alternative to imagine, but most difficult to implement, would be a dedicated keyboard modifier for GEKA, so that it acts more like a keyboard shortcut rather than requiring a mode switch using

**CTRL+ENTER.** This would eliminate one keystroke from each GEKA invocation, leading to increased performance for every command. Another possibility would be to use a quasi-mode for command execution, as Enso does. Rather than having a keystroke to enter GEKA and another keystroke to confirm the command, a single command key would be held down during command selection and released to execute the command. Perhaps both could be provided, leaving the user to choose which to use: a quasi-mode may be most appropriate for zero- and one-parameter commands, and a full mode for multi-parameter commands.

We expect that performance with applications such as Quicksilver and Enso would be roughly comparable to GEKA performance in an experiment similar to the one we ran because many aspects of the interaction are very similar. We did observe, however, that most participants made use of GEKA's built in command abbreviations, which we suspect would give a speed and error rate advantage in comparison to other applications that don't have command abbreviations. Additionally, GEKA's use of a static sorting algorithm likely provides a speed advantage over Quicksilver's adaptive algorithm because once GEKA users have figured out a sequence of keystrokes that will execute a particular command, the mapping will not change. With an adaptive sorting algorithm, the results are unpredictable. The user must visually check the graphical feedback to ensure that the correct command is selected. Finally, many of the commands used in the experiment would not be possible with the more limited command languages used in other applications.

As with any laboratory experiment, the tasks in our study were not fully realistic. Each task was fairly short and consisted entirely of executing commands as opposed to users' typical interactions with word processors that have commands interspersed with extended periods of typing text. Additionally, participants executed several of the commands with methods that they would not generally use, for example using the menu bar to execute

**paste.** A field study where participants use a version of GEKA in their normal working environments could avoid these issues, but it would be at the sacrifice of specific precise measurements that we felt were necessary for a first evaluation.

The long sessions in our laboratory experiment were filled with GEKA and WIMP tasks. There was no time available for interviews or qualitative data collection beyond the questionnaires. Without this richer type of data, we are left with no ability to understand which specific aspects of the GEKA design contributed to users' strong preference for it and which were unsuccessful. Any future studies will have to include more qualitative data collection to help answer these questions.

Our formative study identified dialog boxes as a key WIMP method that needs to be improved upon. Users found keyboard interaction with dialog boxes to be frustrating and nearly always resorted to mouse usage. Unfortunately, our current design has failed to show any significant improvement over dialog boxes. While GEKA multi-parameter commands were rated much better than dialog boxes in our questionnaire and were chosen much more often in the method choice phase of the experiment, there was no quantitative benefit. Speed was shown to be similar, and GEKA had many more errors. Further design iterations of GEKA must focus on providing a performance increase over dialog boxes.

Some limitations of our work have been discussed in previous chapters. Section 3.7 explains the problems with our formative study data being primarily based on self-report rather than actual usage observations. Chapter 4 discusses limitations with the current GEKA design that will need to be addressed in future iterations.

## 6.2 Future work

The previous section describes many limitations of the current GEKA design and of the two studies that we conducted. Future work to address these limitations could include redesigning the GEKA graphical feedback to allow for faster command execution and to better support multiple-parameter commands in order to compete with dialog boxes. Future empirical studies could include a longitudinal study to measure performance over time and a field study to examine GEKA usage in actual work environments. These studies should have a significant qualitative component in order to determine which parts of GEKA work and which don't work, why GEKA is perceived as faster than it actually is, and why some types of commands have far fewer errors than their WIMP counterparts but others have far more.

A major challenge with moving GEKA forward will be expanding beyond one specific application domain. We have demonstrated that the GEKA command language and our prototype graphical feedback work well in the context of word processing. However, we have not yet explored other domains. We envision a consistent GEKA interface being available for all applications and for system-wide global commands, which could pose design challenges.

It is possible that some application domains will contain command structure and object selection methods that force us to rethink our approach to keyboard interaction. For example, some applications might require an object to be selected before a command can be executed, which could conflict with our prefix command language. We believe that expanding to other domains will likely only require small tweaks to our approach, but this is yet to be determined.

Expanding GEKA to support multiple applications will cause issues with command naming. It is likely that two applications will have commands with identical functionality but different names, which could cause confusion

when using GEKA. Perhaps worse would be two applications with commands that have the same name but completely different functionality. An additional area of concern could be global commands that are available regardless of application (such as window manipulation actions). An application attempting to implement a command with the same name as a global command would certainly cause confusion. Each of these issues also applies to choosing a command's short name. Ultimately, it will be up to application developers to decide on the command names and short names that GEKA will use for each specific application. An important piece of research is to develop naming guidelines that will ensure a consistent user experience across applications.

Traditional CLIs have a variety of features beyond basic command executions that help make them very powerful and flexible. History mechanisms help users to find previously executed commands and reuse part or all of the command without the need to input the full command. Aliases and scripting allow users to create custom commands, potentially saving significant time and effort if used frequently. Piping allows data to be easily transferred between applications, which lets multiple small applications be combined in very powerful ways. Each of these features has the potential to be very useful in GEKA, and should be studied.

There are possibilities for GEKA-like interaction to be useful beyond the typical personal computer setup with a standard keyboard and monitor. Tabletops, large scale displays, tablet PCs, and mobile devices all have unique challenges that a GEKA-like approach might help address. Developing versions of GEKA that work with gesture or handwriting recognition might improve the experience of using some of these devices.

GEKA has the potential to be very helpful to people with motor impairments. Some people have a very difficult time using a mouse or other pointing device. Providing these people with a straightforward way to use the keyboard for all applications could be a tremendous benefit to them. It is possible that the GEKA command syntax could even be used with speech

processing to provide an interface for people who cannot use a keyboard. The specific needs of people with motor impairments should be explored in order to create a version of GEKA that will benefit them.

Interesting work could be done in exploring natural-language-like or “sloppy” command syntax as used in Inky (Miller, et al., 2008). For example, there are many ways to phrase the command “insert a table with 4 rows and 2 columns,” including “table rows 4 columns 2,” “table 4x2,” “4x2 table,” “table 4 rows 2 columns,” and “4 row 2 column table.” A future GEKA system could include support for these variations and more. The command variations might be determined by the application designers, through a survey of users eliciting their mental models of commands, or perhaps added to the vocabulary based on users’ “errors” in attempting to input unsupported command variations.



## Bibliography

- Apple Computer. (2009). *Mac Dev Center: Apple Human Interface Guidelines: Introduction to Apple Human Interface Guideleines*. Retrieved 11 23, 2009, from Mac Dev Center: <http://developer.apple.com/mac/library/DOCUMENTATION/UserExperience/Conceptual/AppleHIGuidelines/XHIGIntro/XHIGIntro.html>
- Blacktree. (n.d.). *Blacktree*. Retrieved 11 23, 2009, from <http://www.blacktree.com/>
- Bunt, A., Conati, C., & McGrenere, J. (2007). Supporting interface customization using a mixed-initiative approach. *Proceedings of the 12th international Conference on intelligent User interfaces*, (pp. 92-101).
- Buxton, W. (1982). An Informal Study of Selection-Positioning Tasks. *Proceedings of Graphics Interface '82*, (pp. 323-328). Toronto.
- Cherry, J. M. (1986). An experimental evaluation of prefix and postfix notation in command language syntax. *International Journal of Man-Machine Studies* , 24 (4), 365-374.
- Davison, B., & Hirsh, H. (1997). Toward an adaptive command line interface. *International conference on human-computer interactions No7*.
- Fish Shell. (2009). *Fish Shell*. Retrieved 11 23, 2009, from <http://fishshell.org/>
- Gajos, K., & Weld, D. S. (2004). SUPPLE: automatically generating user interfaces. *Proceedings of the 9th international Conference on intelligent User interfaces* (pp. 93-100). ACM.
- Geller, V. J., & Lesk, M. E. (1983). User interfaces to information systems: choices vs. commands. *Proceedings of the 6th Annual international ACM SIGIR Conference on Research and Development in information Retrieval* (pp. 130-135). New York: ACM.
- Gentner, D., & Nielsen, J. (1996). The Anti-Mac interface. *Commun. ACM* , 39 (8), pp. 70-82.
- Giger, K., & Wilde, E. (2006). XPath filename expansion in a Unix shell. *Proceedings of the 15th international Conference on World Wide Web* (pp. 863-864). ACM.

- Gong, Q., & Salvendy, G. (1995). An approach to the design of a skill adaptive interface. *International Journal of Human-Computer Interaction* , 7 (4), 365-383.
- Greenberg, S., & Witten, I. H. (1988). How users repeat their actions on computers: principles for design of history mechanisms. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM.
- Grossman, T., Dragicevic, P., & Balakrishnan, R. (1997). Strategies for accelerating on-line learning of hotkeys. *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 1591-1600). ACM.
- Grudin, J., & Barnard, P. (1985, April). When does an abbreviation become a word? and related questions. *ACM SIGCHI Bulletin* , 16 (4), pp. 121-125.
- Humanized. (n.d.). *Humanized > Enso*. Retrieved 11 23, 2009, from <http://humanized.com/enso/>
- Karat, J., McDonald, J. E., & Anderson, M. (1986). A comparison of menu selection techniques: touch panel, mouse and keyboard . *International Journal of Man-Machine Studies* , 25 (1), 73-88.
- Lane, D. M., Napier, H. A., Peres, S. C., & Sándor, A. (2005). Hidden Costs of Graphical User Interfaces: Failure to Make the Transition from Menus and Icon Toolbars to Keyboard Shortcuts. *International Journal of Human-Computer Interaction* , 18 (2), 133-144.
- Linton, F., Joy, D., & Schaefer, H. (1999). Building user and expert models by long-term observation of application usage. *Proceedings of the Conference on User Modeling 1999*, (pp. 129-138).
- MacKenzie, I. (1995). Movement time prediction in human-computer interfaces. In R. M. Baecker, W. A. Buxton, J. Grudin, & S. Greenberg, *Readings in human-computer interaction (2nd ed.)* (pp. 483-493). Los Altos, CA, USA: Kaufmann.
- MacLean, A., Carter, K., Löfstrand, L., & and Moran, T. (1990). User-tailorable systems: pressing the issues with buttons. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 175-182). ACM.
- McGrenere, J., Baecker, R. M., & Booth, K. S. (2002). An evaluation of a multiple interface design solution for bloated software. *Proceedings of*

- the SIGCHI Conference on Human Factors in Computing Systems* (pp. 164-170). ACM.
- Microsoft. (2009). *Search Commands*. Retrieved 11 23, 2009, from Microsoft Office Labs:  
<http://www.officelabs.com/projects/searchcommands/Pages/default.aspx>
- Miller, R. C., Chou, V. H., Bernstein, M., Little, G., Van Kleek, M., Karger, D., et al. (2008). Inky: a sloppy command line for the web with rich visual feedback. *Proceedings of the 21st Annual ACM Symposium on User interface Software and Technology* (pp. 131-140). ACM.
- Miller, R., & Myers, B. (2000). Integrating a Command Shell Into a Web Browser. *Proceedings of USENIX 2000 Annual Technical Conference*, (pp. 171-182).
- Mozilla Labs. (2009). *Mozilla Labs >> ubiquity*. Retrieved 11 23, 2009, from <https://mozillalabs.com/ubiquity/>
- Nichols, S., & Ritter, F. E. (1995). A theoretically motivated tool for automatically generating command aliases. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM.
- Norman, D. A. (1988). *The Psychology of Everyday Things*. New York, NY, USA: Basic Books.
- Norman, D. (2007). The next UI breakthrough: command lines. *interactions*, 14 (3), pp. 44-45.
- Peres, S. C., Tamborello, F. P., Fleetwood, M. D., Chung, P., & Paige-Smith, D. L. (2004). Keyboard Shortcut Usage: The Roles of Social Factors and Computer Experience. *Human Factors and Ergonomics Society Annual Meeting Proceedings* (pp. 803-807). Human Factors and Ergonomics Society.
- Raskin, J. (2000). *The Humane Interface*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.
- Shneiderman, B. (1997). *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, 3rd edition. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.
- Shneiderman, B. (1983). Direct Manipulation: A Step Beyond Programming Languages. *IEEE Computer*, 16 (8), 57-69.
- Stuerzlinger, W., Chapuis, O., Phillips, D., & Roussel, N. (2006). User interface façades: towards fully adaptable user interfaces. *Proceedings of*

*the 19th Annual ACM Symposium on User interface Software and Technology*, (pp. 309-318).

Westerman, S. (1997). Individual Differences in the Use of Command Line and Menu Computer Interfaces. *International Journal of Human-Computer Interaction* , 9 (2), 183-198.

Whiteside, J., Jones, S., Levy, P. S., & Wixon, D. (1985). User performance with command, menu, and iconic interfaces. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 185-191). ACM.

Wikipedia. (2009). *Command Line Completion*. Retrieved 11 23, 2009, from Wikipedia - the free encyclopedia:  
[http://en.wikipedia.org/wiki/Command\\_line\\_completion](http://en.wikipedia.org/wiki/Command_line_completion)

YubNub. (n.d.). *YubNub*. Retrieved 12 10, 2009, from <http://yubnub.org/>

## Appendix A Formative Study Documents

Ben Franklin  
[date]  
Page [page]

Oeuf-fondant

Chocolate Rain. Some stay dry and others feel the pain. Chocolate Rain. A baby born will die before the sin. Chocolate Rain. The school books say it can't be here again. Chocolate Rain. The prisons make you wonder where it went. Chocolate Rain. Build a tent and say the world is dry.

Ingredients:

- Egg
- H2O
- Cream
- Chocolate3
- Sulphites

Check Google for more information on how to create this delicious concoction. Additional techniques can also be explored at a place named Wikipedia. If you still need more information or here are some words that don't make sense, you could check out the video site YouTube or look on Facebook. If it's news that you're after: Slashdot or CBC.

Numbers:

14  
16  
18  
20  
22  
24

The secret codes:

a27	b36
c45	d54

Figure A-1: Unformatted document provided to formative study participants.

Ben Franklin  
03/15/09  
Page 1

Oeuf-fondant

Chocolate Rain. Some stay dry and others feel the pain. Chocolate Rain. A baby born will die before the sin. Chocolate Rain. The school books say it can't be here again. Chocolate Rain. The prisons make you wonder where it went. Chocolate Rain. Build a tent and say the world is dry.

Ingredients:

- Egg
- H<sub>2</sub>O
- Cream
- Chocolate<sup>3</sup>
- ~~Sulphites~~

Check [Google](#) for more information on how to create this delicious concoction. Additional techniques can also be explored at a place named [Wikipedia](#). If you still need more information or here are some words that don't make sense, you could check out the video site [YouTube](#) or look on [Facebook](#). If it's news that you're after: [Slashdot](#) or [CBC](#).

Numbers:

14
16
18
20
22
24

The secret codes:

a <sup>2</sup> <sub>7</sub>	b <sup>3</sup> <sub>6</sub>
c <sup>4</sup> <sub>5</sub>	d <sup>5</sup> <sub>4</sub>

Figure A-2: Formatted document for formative study participants to match. The study used a printout with hand written annotations of font sizes, colours, and hyperlink URLs.

## Appendix B   Lab   Experiment   Command Images

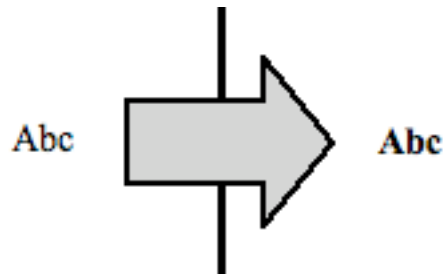


Figure B-1: Lab experiment image for “bold”

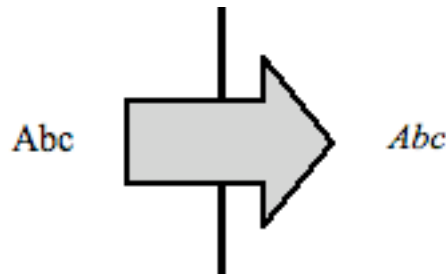


Figure B-2: Lab experiment image for “italic”



Figure B-3: Lab experiment image for “paste”

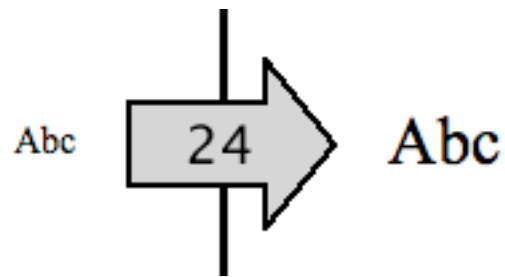


Figure B-4: Lab experiment image for “font size 24”

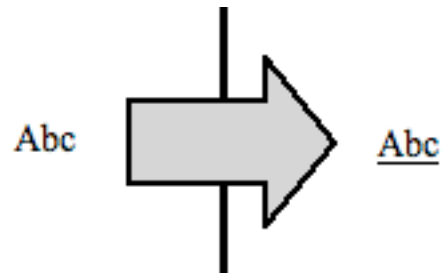


Figure B-5: Lab experiment image for “underline”



Figure B-6: Lab experiment image for “save”



Figure B-7: Lab experiment image for “print copies 3 page range selection”



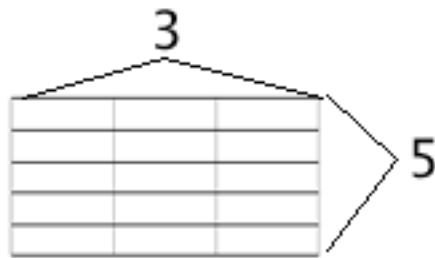


Figure B-8: Lab experiment image for “insert table rows 5 columns 3”

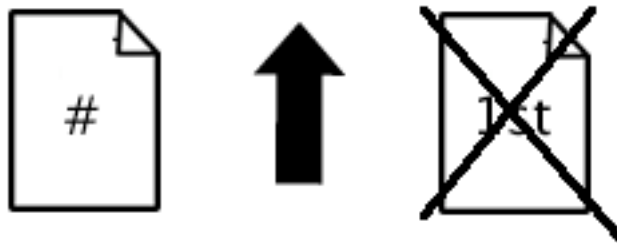


Figure B-9: Lab experiment image for “insert page numbers position top alignment center first page no”

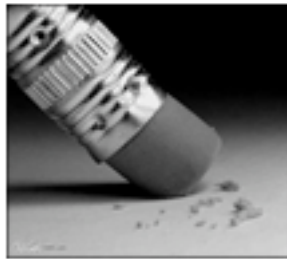


Figure B-10: Lab experiment image for “undo”

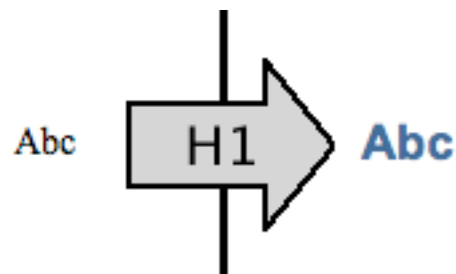


Figure B-11: Lab experiment image for “apply style heading 1”

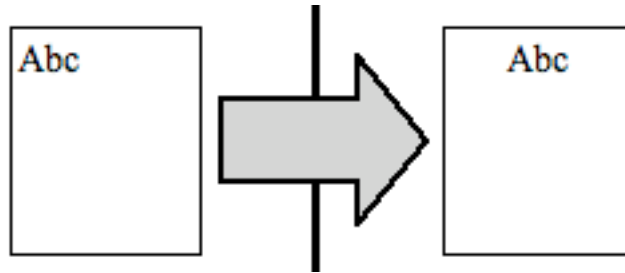


Figure B-12: Lab experiment image for “center”

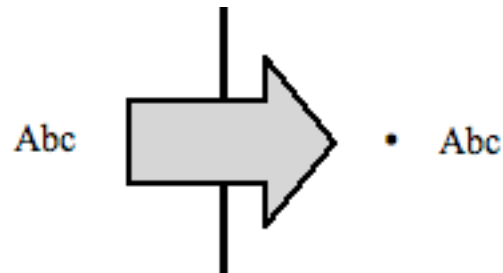


Figure B-13: Lab experiment image for “toggle bullets”

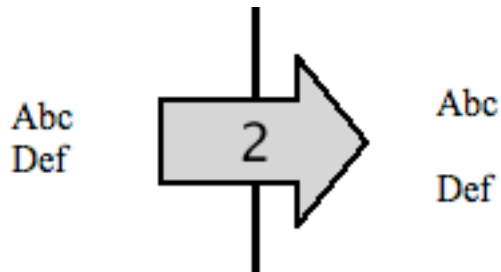


Figure B-14: Lab experiment image for “line spacing 2”











Figure B-15: Lab experiment image for “copy”

## Appendix C Experiment Questionnaire

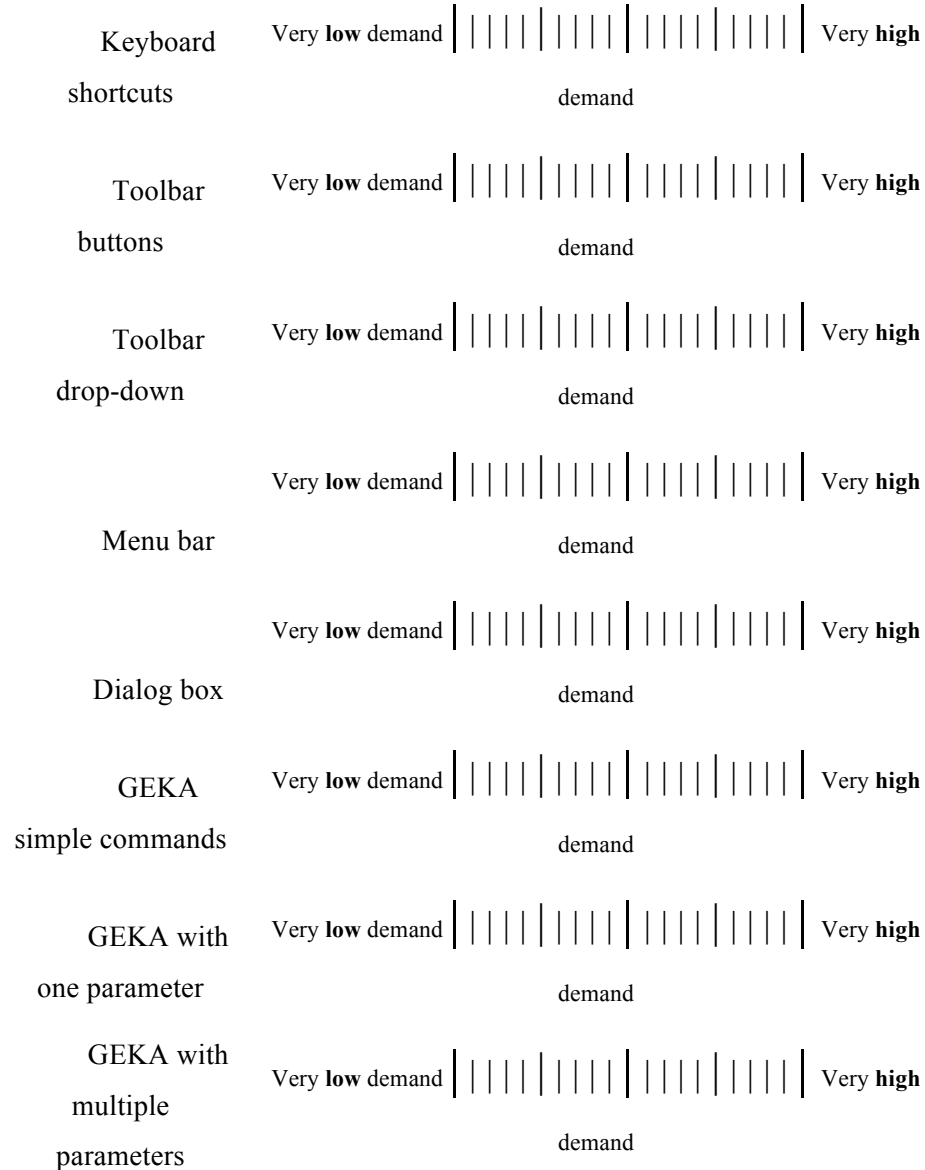
**Easy to learn:** How easy do is it to learn a new command or parameter in each of the following techniques?

Keyboard shortcuts	Very <b>easy</b>		Very <b>hard</b>
Toolbar buttons	Very <b>easy</b>		Very <b>hard</b>
Toolbar drop-down	Very <b>easy</b>		Very <b>hard</b>
Menu bar	Very <b>easy</b>		Very <b>hard</b>
Dialog box	Very <b>easy</b>		Very <b>hard</b>
GEKA simple commands	Very <b>easy</b>		Very <b>hard</b>
GEKA with one parameter	Very <b>easy</b>		Very <b>hard</b>
GEKA with multiple parameters	Very <b>easy</b>		Very <b>hard</b>

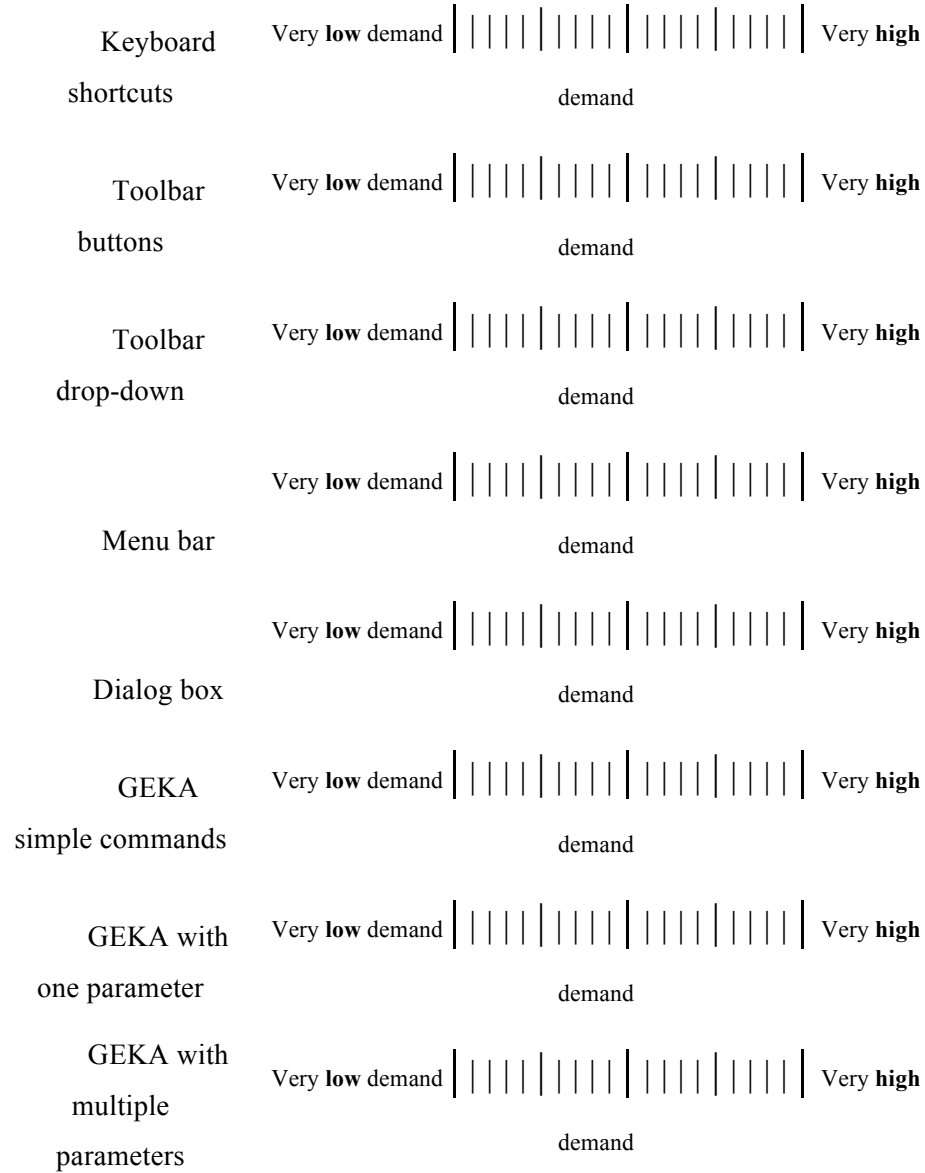
**Easy to remember:** How easy is it to remember how to use a command or parameter that you know in each of the following techniques?

Keyboard shortcuts	Very <b>easy</b>		Very <b>hard</b>
Toolbar buttons	Very <b>easy</b>		Very <b>hard</b>
Toolbar drop-down	Very <b>easy</b>		Very <b>hard</b>
Menu bar	Very <b>easy</b>		Very <b>hard</b>
Dialog box	Very <b>easy</b>		Very <b>hard</b>
GEKA simple commands	Very <b>easy</b>		Very <b>hard</b>
GEKA with one parameter	Very <b>easy</b>		Very <b>hard</b>
GEKA with multiple parameters	Very <b>easy</b>		Very <b>hard</b>

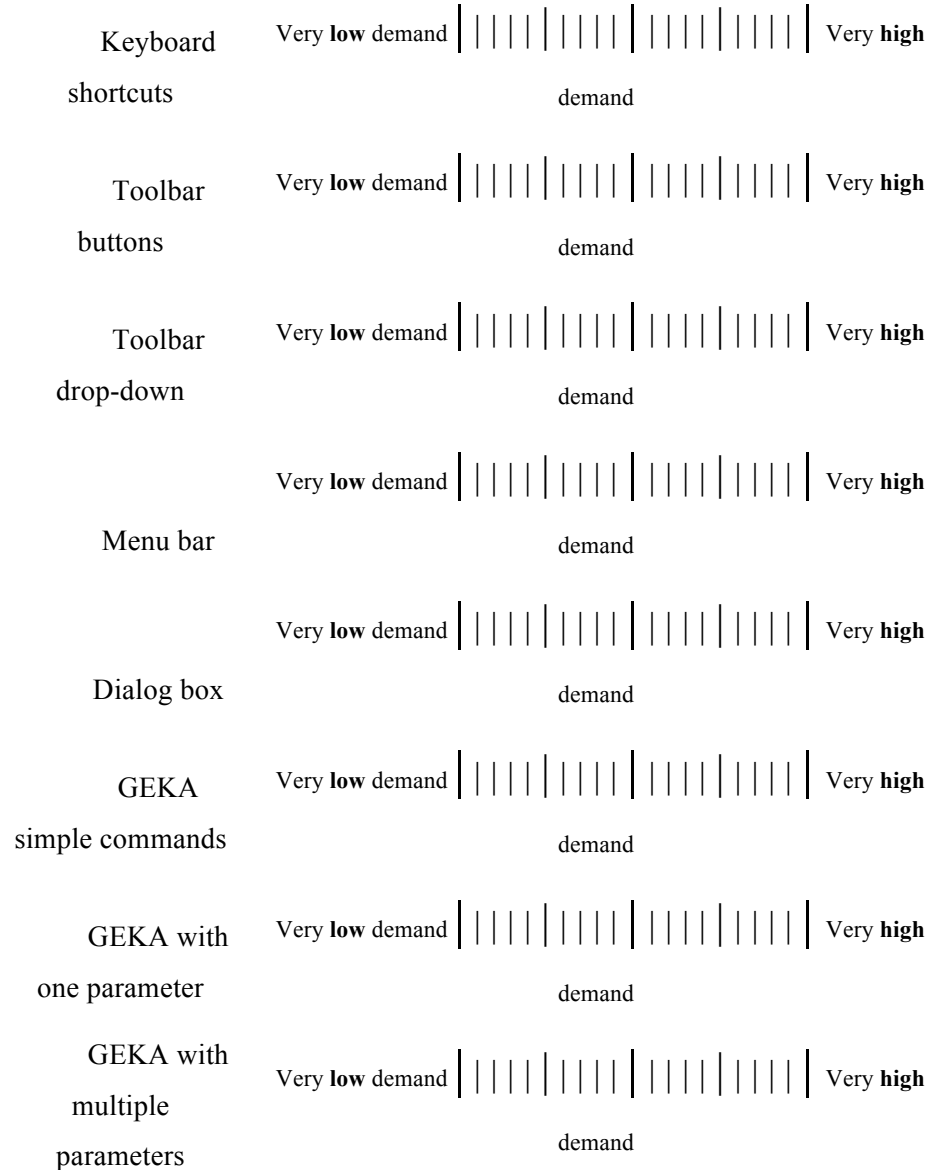
**Mental demand:** How mentally demanding is it to use a command or parameter that you know in each of the following techniques?



**Physical demand:** How physically demanding is it to use a command or parameter that you know in each of the following techniques?



**Visual demand:** How much visual attention is required to use a command or parameter that you know in each of the following techniques?



**Effort:** How hard do you have to work to use a command or parameter that you know in each of the following techniques?

Keyboard shortcuts	Very <b>low</b> effort		Very <b>high</b>
		effort	
Toolbar buttons	Very <b>low</b> effort		Very <b>high</b>
		effort	
Toolbar drop-down	Very <b>low</b> effort		Very <b>high</b>
		effort	
Menu bar	Very <b>low</b> effort		Very <b>high</b>
		effort	
Dialog box	Very <b>low</b> effort		Very <b>high</b>
		effort	
GEKA simple commands	Very <b>low</b> effort		Very <b>high</b>
		effort	
GEKA with one parameter	Very <b>low</b> effort		Very <b>high</b>
		effort	
GEKA with multiple parameters	Very <b>low</b> effort		Very <b>high</b>
		effort	



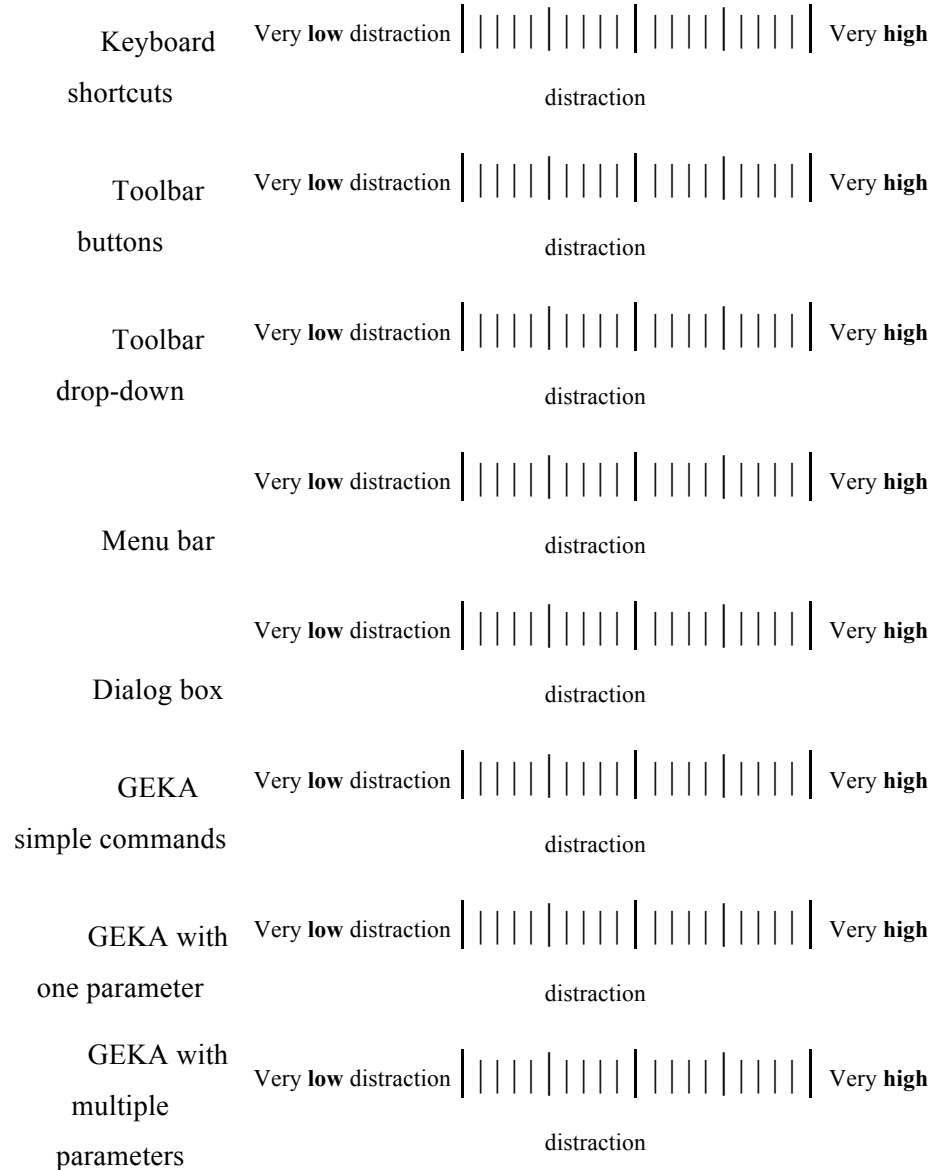
**Tediousness:** How tedious (tiring or boring) is it to use a command or parameter that you know in each of the following techniques?

Keyboard shortcuts	Very <b>low</b> tediousness		Very
		<b>high</b> tediousness	
Toolbar buttons	Very <b>low</b> tediousness		Very
		<b>high</b> tediousness	
Toolbar drop-down	Very <b>low</b> tediousness		Very
		<b>high</b> tediousness	
Menu bar	Very <b>low</b> tediousness		Very
		<b>high</b> tediousness	
Dialog box	Very <b>low</b> tediousness		Very
		<b>high</b> tediousness	
GEKA simple commands	Very <b>low</b> tediousness		Very
		<b>high</b> tediousness	
GEKA with one parameter	Very <b>low</b> tediousness		Very
		<b>high</b> tediousness	
GEKA with multiple parameters	Very <b>low</b> tediousness		Very
		<b>high</b> tediousness	

**Frustration:** How frustrating (discouraging, irritating, or annoying) is it to use a command or parameter that you know in each of the following techniques?

Keyboard shortcuts	Very <b>low</b> frustration		Very <b>high</b> frustration
Toolbar buttons	Very <b>low</b> frustration		Very <b>high</b> frustration
Toolbar drop-down	Very <b>low</b> frustration		Very <b>high</b> frustration
Menu bar	Very <b>low</b> frustration		Very <b>high</b> frustration
Dialog box	Very <b>low</b> frustration		Very <b>high</b> frustration
GEKA simple commands	Very <b>low</b> frustration		Very <b>high</b> frustration
GEKA with one parameter	Very <b>low</b> frustration		Very <b>high</b> frustration
GEKA with multiple parameters	Very <b>low</b> frustration		Very <b>high</b> frustration

**Distraction:** How much does it distract you from your main task to use a command or parameter that you know in each of the following techniques?



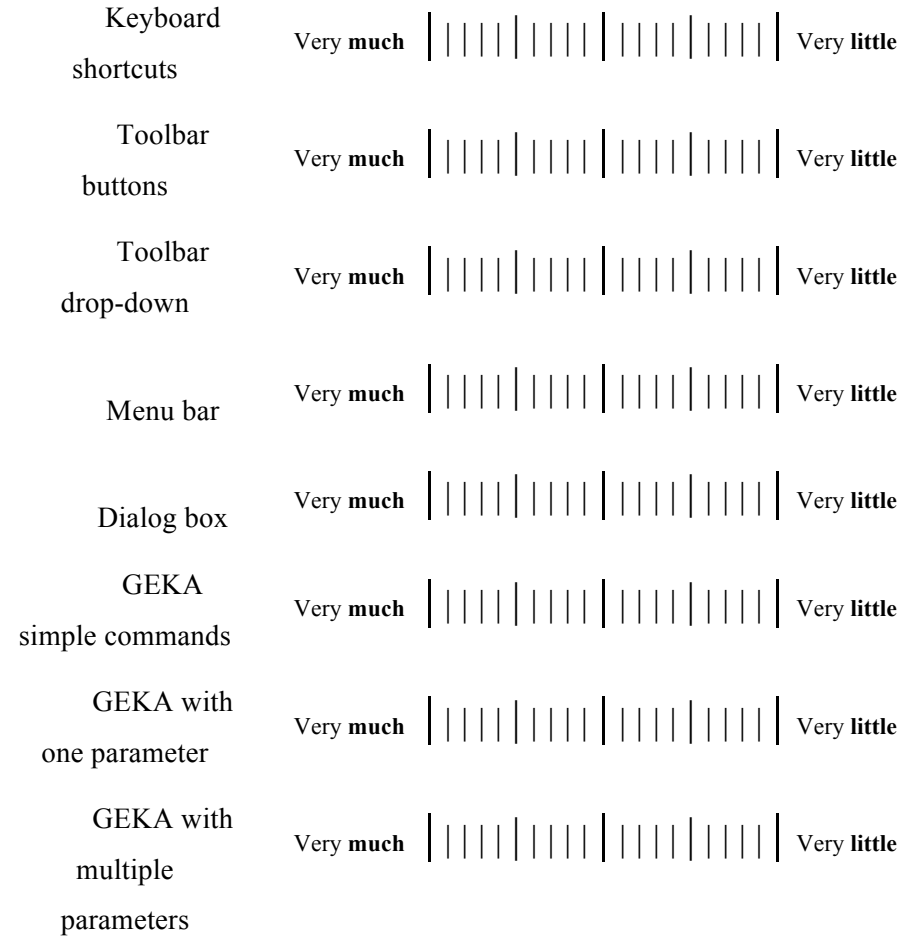
**Speed:** How fast is it to use a command or parameter that you know in each of the following techniques?

Keyboard shortcuts	Very <b>fast</b>		Very <b>slow</b>
Toolbar buttons	Very <b>fast</b>		Very <b>slow</b>
Toolbar drop-down	Very <b>fast</b>		Very <b>slow</b>
Menu bar	Very <b>fast</b>		Very <b>slow</b>
Dialog box	Very <b>fast</b>		Very <b>slow</b>
GEKA simple commands	Very <b>fast</b>		Very <b>slow</b>
GEKA with one parameter	Very <b>fast</b>		Very <b>slow</b>
GEKA with multiple parameters	Very <b>fast</b>		Very <b>slow</b>

**Error rate:** How many errors do you make when using a command or parameter that you know in each of the following techniques?

Keyboard shortcuts	Very few errors	<div> <div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div> <div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div> </div>	Very
		many errors	
Toolbar buttons	Very few errors	<div> <div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div> <div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div> </div>	Very
		many errors	
Toolbar drop-down	Very few errors	<div> <div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div> <div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div> </div>	Very
		many errors	
Menu bar	Very few errors	<div> <div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div> <div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div> </div>	Very
		many errors	
Dialog box	Very few errors	<div> <div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div> <div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div> </div>	Very
		many errors	
GEKA simple commands	Very few errors	<div> <div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div> <div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div> </div>	Very
		many errors	
GEKA with one parameter	Very few errors	<div> <div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div> <div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div> </div>	Very
		many errors	
GEKA with multiple parameters	Very few errors	<div> <div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div> <div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div> </div>	Very
		many errors	

**Overall opinion:** How much to you enjoy using each of the following techniques?



## **Appendix D   Ethics Approval Certificates**



The University of British Columbia  
Office of Research Services  
**Behavioural Research Ethics Board**  
Suite 102, 6190 Agronomy Road, Vancouver, B.C. V6T 1Z3

## CERTIFICATE OF APPROVAL - MINIMAL RISK

<b>PRINCIPAL INVESTIGATOR:</b> Joanna McGrenere	<b>INSTITUTION / DEPARTMENT:</b> UBC/Science/Computer Science	<b>UBC BREB NUMBER:</b> H08-02562
<b>INSTITUTION(S) WHERE RESEARCH WILL BE CARRIED OUT:</b>		
<b>Institution</b> UBC Other locations where the research will be conducted: none		<b>Site</b> Vancouver (excludes UBC Hospital)
<b>CO-INVESTIGATOR(S):</b> Jeffrey C. Hendy Kellogg S. Booth		
<b>SPONSORING AGENCIES:</b> Natural Sciences and Engineering Research Council of Canada (NSERC)		
<b>PROJECT TITLE:</b> Investigating the Graphical Command Line		
<b>CERTIFICATE EXPIRY DATE:</b> March 13, 2010		
<b>DOCUMENTS INCLUDED IN THIS APPROVAL:</b>		<b>DATE APPROVED:</b> March 13, 2009
<b>Document Name</b>	<b>Version</b>	<b>Date</b>
<b>Consent Forms:</b> Consent Form	N/A	January 15, 2009
<b>Advertisements:</b> Email poster	N/A N/A	January 15, 2009 February 28, 2009
<b>Questionnaire, Questionnaire Cover Letter, Tests:</b> Interview scripts	N/A	January 29, 2009
<b>Other:</b> none		
The application for ethical review and the document(s) listed above have been reviewed and the procedures were found to be acceptable on ethical grounds for research involving human subjects.		
Approval is issued on behalf of the Behavioural Research Ethics Board and signed electronically by one of the following:  Dr. M. Judith Lynam, Chair Dr. Ken Craig, Chair Dr. Jim Rupert, Associate Chair Dr. Laurie Ford, Associate Chair Dr. Anita Ho, Associate Chair		





The University of British Columbia  
Office of Research Services  
**Behavioural Research Ethics Board**  
Suite 102, 6190 Agronomy Road, Vancouver, B.C. V6T 1Z3

## CERTIFICATE OF APPROVAL - MINIMAL RISK AMENDMENT

<b>PRINCIPAL INVESTIGATOR:</b> Joanna McGrenere	<b>DEPARTMENT:</b> UBC/Science/Computer Science	<b>UBC BREB NUMBER:</b> H08-02562
<b>INSTITUTION(S) WHERE RESEARCH WILL BE CARRIED OUT:</b>		
<b>Institution</b> UBC Other locations where the research will be conducted: none		<b>Site</b> Vancouver (excludes UBC Hospital)
<b>CO-INVESTIGATOR(S):</b> Jeffrey C. Hendy Kellogg S. Booth		
<b>SPONSORING AGENCIES:</b> Natural Sciences and Engineering Research Council of Canada (NSERC)		
<b>PROJECT TITLE:</b> Investigating the Graphical Command Line		

Expiry Date - Approval of an amendment does not change the expiry date on the current UBC BREB approval of this study.  
An application for renewal is required on or before: **March 13, 2010**

<b>AMENDMENT(S):</b>	<b>AMENDMENT APPROVAL DATE:</b> June 17, 2009	
<b>Document Name</b>	<b>Version</b>	<b>Date</b>
<b>Advertisements:</b>		
PHASE 2 - poster	N/A	June 3, 2009
PHASE 2 - email	N/A	June 3, 2009
<b>Questionnaire, Questionnaire Cover Letter, Tests:</b>		
PHASE 2 - interface comparison questionnaire	N/A	June 5, 2009
<b>Other:</b> none		
The amendment(s) and the document(s) listed above have been reviewed and the procedures were found to be acceptable on ethical grounds for research involving human subjects.		
Approval is issued on behalf of the Behavioural Research Ethics Board		
Dr. M. Judith Lynskey, Chair Dr. Ken Craig, Chair Dr. Jim Rupert, Associate Chair Dr. Laurie Ford, Associate Chair Dr. Anita Ho, Associate Chair		