Graphically Enhanced Keyboard Accelerators for GUIs

Jeff Hendy Kellogg S. Booth Joanna McGrenere

Department of Computer Science
The University of British Columbia
{jchendy, ksbooth, joanna}@cs.ubc.ca

ABSTRACT

We introduce GEKA, a graphically enhanced keyboard accelerator method that provides the advantages of a traditional command line interface within a GUI environment, thus avoiding the "Fitts-induced bottleneck" of pointer movement that is characteristic of most WIMP methods. Our design rationale and prototype development were derived from a small formative user study, which suggested that advanced users would like alternatives to WIMP methods in GUIs. The results of a controlled experiment show that GEKA performs well, is faster than menu selection, and is strongly preferred over all mouse-based WIMP methods.

KEYWORDS: command line, GUI, shortcut, WIMP, experiment

INDEX TERMS: H.5.2 [Information interfaces and presentation]: User Interfaces

1 INTRODUCTION AND MOTIVATION

The graphical user interface (GUI) was developed to overcome limitations of the once-dominant command line interface (CLI). An emphasis on recognition rather than recall, afforded by visible representations of commands and their parameters, makes GUI interfaces easier to learn and less error prone. The rich graphical feedback that GUIs provide also makes errors easier to detect and correct. As applications became increasingly sophisticated, every major operating system came to provide a windows-icons-menus-pointer (WIMP) interface. But WIMP interaction is not without problems. WIMP interfaces use a pointing device and interaction methods such as menus, toolbars, and direct manipulation actions. The focus in designing pointer-based interaction methods is on making actions easy to learn and remember. While helpful for new or infrequent users, these methods fall short for advanced, frequent users who want to execute commands quickly with little distraction from their main task.

GUIs have introduced keyboard shortcuts to help advanced users execute commands quickly. Shortcuts have several limitations of their own. When the number of commands grows beyond a handful, shortcuts end up being complicated and bear little resemblance to the command name. When applications have dozens of obscure key combinations, it becomes very difficult to learn and remember all of them. Furthermore, the set of shortcuts is typically incomplete – shortcuts are generally not available for every command – and they provide no support for parameters. Some applications include support for custom keyboard shortcuts. This partially addresses the issue of incompleteness, but the number of useful shortcuts is still limited by the number of keys on the keyboard and the user's memory.

The limitations of WIMP interaction described above are well known and described in many books [12] [13]. In 1996, Gentner and Neilsen [2] described many problems with GUIs saying that "direct manipulation quickly becomes repetitive drudgery" and that see-and-point interfaces behave "as if we have […] lost our facility with expressive language, and been reduced to pointing at objects in the immediate environment." They recommend a focus on expert users and an interface based on language.

In 2000, Raskin [11] 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." In 2007, Norman [10] predicted that one of the next "UI breakthroughs" will be related to command lines, stating that "GUIs work well only when the number of alternative items or actions is small."

1.1 Goals of the Research

Most GUIs now offer a choice between two very distinct alternatives: pointer-based interaction that focuses mostly on ease of use and keyboard-based interaction that focuses mostly on speed of use. We see an empty design space between these two extremes that we call the "GUI gap." Our goal is to fill this gap with a novel *graphically enhanced keyboard accelerator* (GEKA) interaction method that makes use of CLI-like syntax augmented by an incremental search mechanism with graphical feedback to quickly and easily select desired commands and parameters. GEKA bridges the GUI gap by making appropriate compromises, resulting in interaction that is reasonably fast, available for most commands, and supports parameters, yet is still straightforward to learn and use because it supports recognition rather than recall.

There are three contributions in this paper. The first is preliminary empirical data on usage patterns of different GUI methods and qualitative feedback we obtained about user dissatisfaction with dialog boxes. The second is the design of the GEKA command language and a prototype implementation. The third is experimental validation showing that users can use GEKA quickly, with no overall increase in errors, and that GEKA is strongly preferred over mouse-based methods.

2 DESIGN RATIONALE AND OBJECTIVES

GEKA is designed to augment WIMP interfaces by allowing most commands to be executed through the keyboard, thus providing a more efficient and satisfying experience for advanced users. We do not expect novices or infrequent users to benefit as much from GEKA, although we do not intend to disadvantage them or preclude incremental adoption as users gain experience. Our initial work in GEKA was approached with several concrete goals derived from these objectives:

Speed – In order to be attractive to advanced users, GEKA should be faster than pointer-based WIMP methods. We recognize that GEKA will be slower than current keyboard shortcuts. We thus expect continued use of keyboard shortcuts where appropriate, but believe that as long as GEKA is noticeably faster than pointer-based methods, it will be well received.

Low errors – GEKA should not be more error prone than mouse-based methods. A frequent need to correct errors would

Graphics Interface Conference 2010
31 May - 2 June, Ottawa, Ontario, Canada

negate the benefit of faster command execution. We want GEKA to have error rates similar to existing methods.

Easy to learn and remember – We want GEKA to be easier to learn and remember than keyboard shortcuts. Making extensive use of graphical feedback, the basis of GEKA interaction is recognition rather than recall, eliminating the need to memorize obscure keystroke combinations.

Low visual demand – One huge drawback of pointer-based interaction methods is that the user's visual attention has to be fixed on the pointer to ensure that the right item is being selected. GEKA is designed to allow practiced users to execute commands with little or no visual attention.

Completeness and choice – GEKA should be available for use with most commands and parameters. While we don't expect all users, or even all advanced users, to use GEKA every time they execute a command, it is important to give users a choice. Allowing most commands to be executed with either the mouse or the keyboard could relieve a lot of frustration where users are currently forced to move their hands from the keyboard to the mouse just to execute one command.

3 Related Work

We review other GUI enhancements, discuss the influence of CLIs on our work, and highlight relevant empirical studies.

3.1 Applications that offer CLI alternatives to WIMP

Quicksilver [1] 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 from its catalog. 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. For example, 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.

Enso [4] 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 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. A Firefox plug-in called Ubiquity [9] was introduced while our design work was underway. Ubiquity offers interaction similar to Enso but with support for multiple parameters in a fixed order.

Inky [8] 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.

Built-in OS and application support for CLI alternatives – Mac OS X 10.5 has a search box in the help menu that locates all menu items in the current application that match the input. There is a plug-in available for Microsoft Office 2007 [7] that has similar functionality to locate commands in the ribbon. These two alternatives are more focused on locating items within the GUI than actually executing commands: they search only the top-level menu or ribbon items and have no support for parameters. Modern IDEs, including Eclipse and Visual Studio, have well developed auto-complete features that help search for possible variable or function names and allow easy input of parameters.

3.2 Traditional command line interfaces

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 shorter specifications because parameters whose default values were appropriate need not be listed. The original command completion feature on the SDS 940 Genie operating system was automatic – as soon as the stem uniquely determined a command the full command name was typed by the system. This was later modified in the PDP-10 Tenex CLI so command completion only took place when **`ESC`** was typed and this was extended to provide file name completion. This led to **`TAB`** completion in Unix **`tcsh`**, which provides a list of possible completions as recognition-based hints to the user if **`CTRL-D`** is typed instead of **`TAB`** [14].

There is a clear pattern. 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. Many of these same ideas apply to GUIs.

3.3 Empirical studies of keyboard accelerators

Attempts to utilize keyboard-based interaction within GUIs are not new. The most obvious example is keyboard shortcuts. Recent work by Lane et al. [5] found that use of keyboard shortcuts was low in a survey of experienced Microsoft Word users, and Grossman et al. [3] explored ways to help users learn and use keyboard shortcuts more effectively. We were surprised by the very low shortcut usage in Lane et al.'s study, which does not match our perception of experienced users. Our work builds on this study by focusing on highly advanced computer users.

4 Formative Study

To verify that a new keyboard-based interaction technique would benefit advanced GUI users, and to validate our GEKA goals, we conducted a small formative study to examine advanced computer users' current habits. We were interested in comparing Lane et al.'s [5] results of low shortcut usage to a more experienced group of participants. We were also interested in exploring how advanced users interact with dialog boxes and gathering qualitative feedback on when and why users choose each interaction method.

Consistent with our goal to improve interaction for advanced users, our 10 participants (3 female) included nine computer science graduate students and one computer engineering graduate student. In the study, we worked with Microsoft Word 2003, with which all participants indicated a high level of familiarity.

Participants were interviewed about their use of 26 word processor commands, selected based on Linton, Joy, and Schaefer's 1999 list of most frequently used commands [6], with some no-longer-used commands removed and a few common formatting commands added. Participants were asked how often they use each command, which method (shortcut, drop-down menu, context menu, or toolbar) they most frequently use to execute the command, which other methods they sometimes use, and which methods they know but don't use. The results of this

	Paste	Save	Copy	Bold	Cut	Undo	Underline	New File	Find	Superscript	Redo	Close Doc	Quit App	Center	Font	Size	Style	Font Color	Zoom	Open	Print	Save as	Print Pre	Head	Table	Insert Pic	Avg*
Frequently	9	9	9	6	7	7	5	7	6	0	4	6	8	3	7	7	1	1	3	6	6	5	4	0	0	0	13
Sometimes	1	1	1	3	2	3	2	1	4	5	2	3	1	4	2	3	6	4	4	3	4	3	4	4	6	5	8
Rarely	0	0	0	1	1	0	3	2	0	5	4	1	1	3	1	0	3	5	3	1	0	2	2	6	3	4	5
Never	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
Know KB Shortcut	10	10	10	10	9	9	10	7	9	5	5	4	4	1	1	2	0	0	0	8	5	1	0	0	0	0	12
Menu	0	0	0	0	0	2	0	3	2	4	3	1	0	0	0	0	1	0	1	5	8	8	7	8	6	7	7
KB Short	10	9	10	9	8	9	7	6	8	5	5	3	3	1	0	2	0	0	0	3	2	1	0	0	0	0	10
Context menu	0	0	0	0	2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Toolbar	0	1	0	2	0	0	4	2	1	0	2	7	8	9	10	8	9	10	9	3	1	0	2	1	3	1	9

Figure 1: Command usage in Word 2003. The first four rows show number of participants who use each command with each frequency. The fifth row shows number of participants who know keyboard shortcuts for each command. The final four rows show number of participants who use each method most frequently for each command. (* The final column is the average number of commands in each category per participant, i.e. the sum of the columns divided by 10 participants) (*N* = 10).

part of the study are summarized in Figure 1.

Participants were next asked to consider commands that they use frequently and know how to execute using all four methods, and then to order the methods in terms of preference. All participants ranked keyboard shortcuts first and drop-down menus last, with toolbars and context menus varying between second and third place. Common reasons for liking keyboard shortcuts included speed, precision, and being able to keep one's hands in the same place. Common reasons for disliking menus included the need for multiple clicks and scanning through many options.

While our participants stated a strong preference for keyboard shortcuts and reported far more shortcut usage than did the less experienced users studied by Lane et al. [5], shortcuts still had a fairly low usage. The bottom portion of Figure 1 shows how many users selected each method as most frequently used for each command. Shortcuts are the most commonly used method for only 11 of the 26 commands, even though 21 of the 26 commands have shortcuts available. There were 16 commands where at least half of the participants did not even know the keyboard shortcut.

Figure 2 shows how often each method is used for each frequency of command usage. As we expected, keyboard shortcuts are the most used method for frequently used commands. Even so, mouse-based methods are used for nearly half of the frequently used commands. For sometimes-used and rarely-used commands, mouse-based methods dominate.

When asked why they would use mouse-based methods when they preferred keyboard shortcuts, all participants said that the main reason is "not knowing the shortcut." Other reasons include having their hand already on the mouse, and "habit" from when they first learned the command.

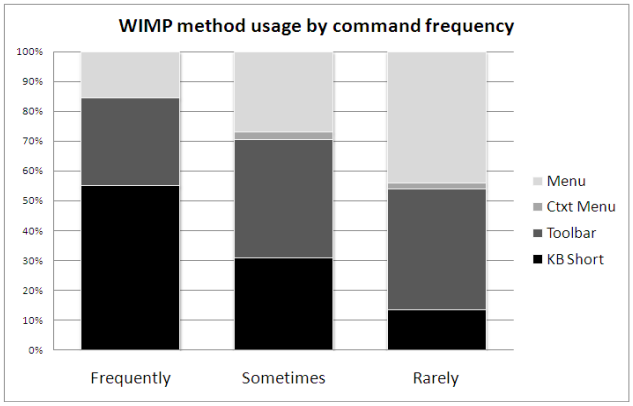

Figure 2: Percentage of commands reported to be most frequently executed with each technique broken down by frequency of command use (*N*=10).

The final part of our formative study examined dialog box usage to expose any issues with multiple-parameter commands in WIMP interaction. Participants were observed completing tasks with dialog box usage. Across the 10 participants, there were 40 total command executions that could have been completed using the keyboard exclusively. Keyboard shortcuts were used 16 times to invoke the dialog box. Ten of those 16 times, the participant moved their hand to the mouse at some point to navigate in the dialog box. When asked why they would switch to the mouse to navigate dialog boxes, participants said that using the keyboard to navigate through a dialog box is unpredictable and slow because they aren't sure where the cursor will move when the **TAB** key is pressed. This suggests a significant shortcoming with dialog boxes, as even when a command was started with the keyboard, participants switched to the mouse more than half the time.

4.1 Discussion

Advanced users prefer keyboard interaction – At least while word processing, where users spend most of the time with hands on the keyboard, our users indicated that they like to use keyboard shortcuts as often as possible to speed up command execution.

Keyboard shortcuts are insufficient – While our advanced computer users did report more shortcut usage than Lane et al.'s participants [5], usage was still fairly low, mainly due to obstacles in learning and remembering the shortcuts.

Keyboard navigation in dialog boxes is insufficient – When faced with a dialog box, most of our users relied on the mouse to select parameters due to the confusing nature of keyboard navigation in dialog boxes.

Filling the GUI gap – While our formative study was limited in terms of the number of participants and the reliance on self-reported data, it certainly suggests that advanced computer users are in fact suffering the consequences of the GUI gap and motivates our work on GEKA. Our advanced users wanted quick keyboard interaction for executing commands, but were too often forced to resort to mouse-based methods. If GEKA can successfully fill this gap, these users would clearly benefit.

5 GEKA Design

We envision future iterations of GEKA-style interaction being available for most commands in most applications. However, in order to maintain a reasonable scope for our initial design efforts, we chose to work only within a specific application domain. We chose word processors because they tend to exemplify WIMP

interfaces and because they have a rich enough command set to illustrate all of the issues we want to address. For our laboratory experiment, described below, we created a replica of the Microsoft Word 2003 user interface. Our prototype runs within that context.

Our initial focus has been on the fundamentals of GEKA interaction, a command language, and a command and parameter completion algorithm. We also implemented a graphical feedback component to support recognition over recall.

5.1 Command language and auto-completion

GEKA works as a separate command mode within the application, entered by typing **`CTRL+ENTER`**. The GEKA command syntax resembles a traditional CLI. A command name is selected, after which optional parameters may be specified by selecting a parameter name followed by a parameter value. Examples include:

`bold` - *no parameters*

`zoom 200` - *only one possible parameter*

`print pages 1-5` - *one parameter used of many possibilities*

`insert_table rows 2 columns 3` - *multiple parameters*

As each character is typed, an incremental search mechanism lists all possible matches and selects a most likely match. These are presented in the graphical feedback component. This eliminates the need to type or memorize full command or parameter names. Each command has a "short name," an abbreviated sequence of characters that will always select that command as the first match. In our initial prototype, short names are typically between 1 and 3 characters and are based on one of the following: a command's keyboard shortcut, the first letters of each word of a command's name, or the first few letters of one word in a command name. This allows frequently used commands to be executed quickly without having to examine the graphical feedback if a user knows the short name.

The "match list" for auto completion is ordered in a way that should place the intended command near the top. There are four categories of matches, with all results that fall under one category occurring in the list before the next category begins. Results within each category of match are sorted alphabetically. An example of a match list is shown in Figure 3. The four categories of matches are as follows.

Exact match – the entered text is exactly the command or parameter name (either the full name or the short name)

Prefix match – the command or parameter name begins with the characters in the entered text

Substring match – the entered text is a contiguous sequence within the command or parameter name

Subsequence match – each character in the entered text is contained in the command or parameter name in order, but there may be other characters in between

5.2 Graphical feedback

We implemented a simple graphical feedback component. It is based on the three-panel layout of Quicksilver, which fits nicely with the three distinct phases of entering a command in GEKA (command name, parameter name, parameter value). However, the underlying interaction in GEKA is very different from Quicksilver's structure (object search, action name, parameter).

Our prototype is shown in Figure 3. Part A shows the initial state when opening GEKA, which lists all commands alphabetically. As the user types each character, the match list is refined and the best match is visible at the top of the pane, replacing the initial prompt (parts B, C, and D of Figure 3). The best matching command name shows which characters in the name match the input by making them red. It also shows the command's short name by underlining those characters. In part B, the command has been selected by typing its short name '**`p`**' and thus that character is both red and underlined.

When the best matching command is one that has parameters, a second pane appears (part B of Figure 3), listing the parameters and their current values. Pressing the space bar will move input focus to the second panel and allow selection of parameters.

For commands with only one parameter, the value can be entered right away (part C of Figure 3). For commands with multiple parameters, a parameter name must first be selected. This is done by typing some, or all, of the parameter name. The list of parameters and the best matching parameter are determined just as for the command list. Once a parameter name is selected, pressing the space bar again allows a value to be entered. Text and numerical values are simply typed in, while choices from a list of possible values are handled with an incremental search just like command selection. After a parameter value has been selected, pressing the space bar again moves the focus back to the second

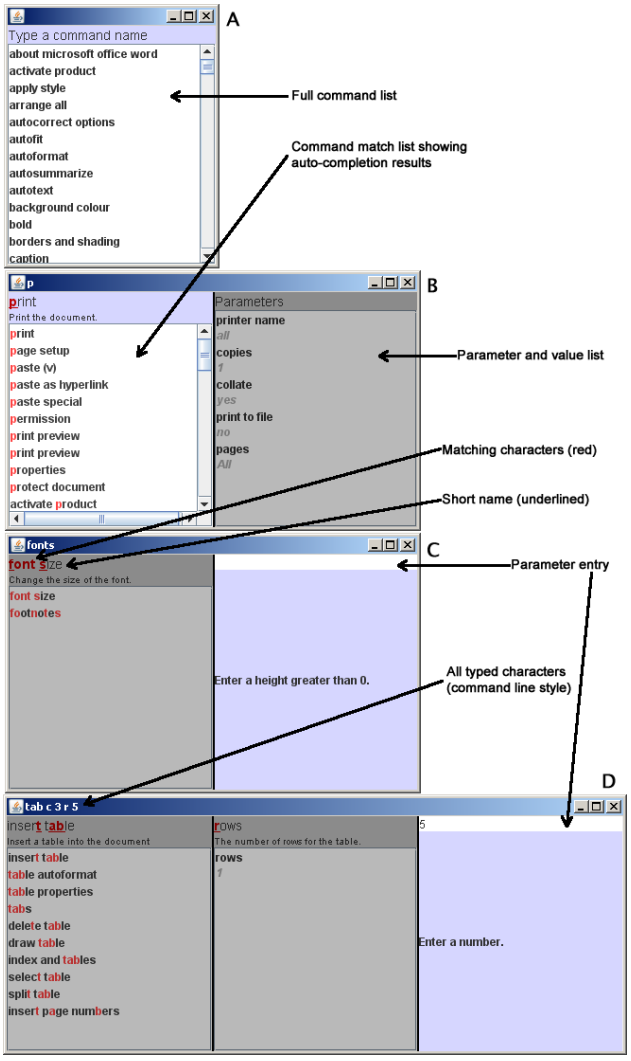

Figure 3: GEKA prototype graphical feedback showing four distinct phases of interaction. Part A is before anything has been typed, B shows command name selection, C shows inputting a parameter for a command with only one parameter, and D shows inputting a parameter for a command with multiple parameters.

pane. Our prototype does not allow spaces in command or parameter names.

The title bar of the window shows all of the characters that have been entered, as shown in part D of Figure 3 where the table command has been selected after "**tab c 3 r 5**" has been entered. This allows an advanced user executing a known command to use GEKA just like a traditional command line without needing to look at any of the rest of the feedback.

Pressing enter at any time will execute the best matching command with its selected parameters. The backspace key will clear the most recently entered character and, if appropriate, change the state of the graphical feedback window to reflect the change. If, in part C of Figure 3, backspace were pressed, the space, which was what moved focus to the second GEKA pane, would be eliminated from the end of the input string "fonts ", and thus focus would return to the first (leftmost) GEKA pane for command selection. Pressing the escape key exits GEKA without executing a command.

GEKA has three key design improvements over existing applications: (1) support for multiple parameters in arbitrary order, (2) smarter matching including abbreviations for all commands, (3) clear visual feedback of the input characters to facilitate learning and re-use. GEKA's support for an essentially unrestrictedly large command vocabulary offers a clear advantage over keyboard shortcuts, which are limited by the number of keys and modifiers on the keyboard.

5.3 Examining the command language

To assess whether our goal of executing most commands through GEKA is realistic, we examined the actions available in Microsoft Word 2003 and made note of the cases where GEKA might be problematic:

Multi-part dialogs between the user and the computer – Completing actions that require multiple iterations of user input, such as find/replace, spell check, or any action involving a wizard, will require extending GEKA.

Inherently visual tasks – Actions that rely heavily on graphical representation of parameters, such as selecting a colour or inserting special symbols, could be challenging for novice or infrequent users unless new naming conventions or a preview capability are introduced.

Direct manipulation – Operations such as the format paintbrush require text to be "painted" with the mouse. This could be adapted to GEKA by decomposing it into copy format and paste format components with selections performed independently of the command.

Nested or multi-part parameters – Some parameters are not simple name-value pairs. When sorting values in a table, the separate fields at parameter has a value other which requires further input of the actual value. The rigid three-panel layout of our graphical feedback would need to be extended for this.

GEKA's command language is robust, and most of these issues can be addressed by redesigning just the graphical feedback component. Instead of the existing rigid structure for displaying parameters, each command could have its own specifically tailored visual feedback. This might resemble current dialog boxes, but be controlled through the GEKA command language rather than the primitive **TAB**-based method now in use. Future research will determine how to best resolve these issues.

6 Laboratory Experiment

We conducted a laboratory experiment with our GEKA prototype to explore how well users can learn and use GEKA, how their performance in GEKA compares to WIMP performance, and whether they will use GEKA when given a choice.

6.1 Experimental tasks

We created a replica of the Microsoft Word 2003 user interface using the C# programming language. Our software has the same toolbar and menu layout as Word 2003. Dialog boxes were recreated where needed. Our GEKA prototype is programmed in Java and communicates with the Word replica through standard input/output redirection.

In this study, we worked with a slightly different set of WIMP methods than in the formative study. Toolbars were further categorized as either buttons or drop-downs for direct comparison to zero- and one-parameter GEKA commands. Dialog boxes were added to compare to multiple-parameter GEKA commands. Context menus were not considered because the formative study showed that they are rarely used.

To compare GEKA to each of these WIMP methods, we chose three commands from each category to test with users. Each command is representative of its method and should be familiar to advanced Microsoft Word users. We implemented the following 15 commands.

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 numbers

In addition to the above commands, which were fully implemented in both the WIMP interface and in GEKA, the WIMP interface contained all of the menu items and toolbar buttons from Word 2003, and the GEKA command list contained all of those same items. Commands not listed above, however, had no functionality in our prototype.

In the experimental environment (see Figure 4), there is an instructions window on the left side of the Word replica screen to instruct the participant on which commands to execute. Each screen of instructions is considered as one task, and is composed of a text selection at the top followed by four command specifications below. Because prompting participants with the command name could bias performance in GEKA, we used images to represent each command. The three images shown on the left in the screenshot in Figure 4 are typical. They represent insert_page_numbers position top alignment center first_page no,

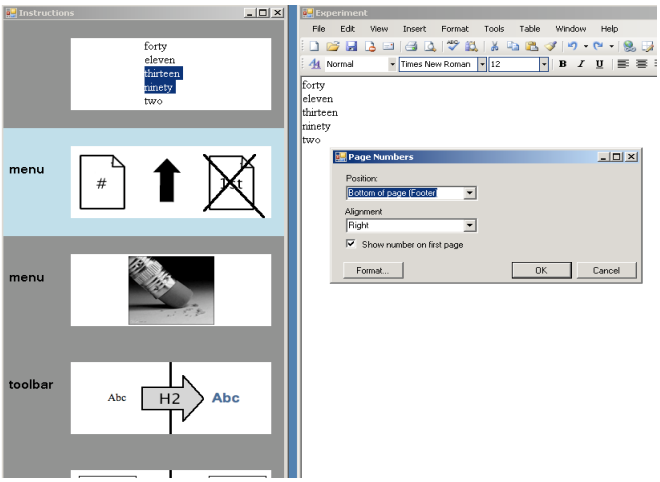

Figure 4: Screenshot of the experimental environment.

undo, and apply_style heading_2. A fourth image for center alignment is below the others. The command highlighted in blue represents the current command (insert page numbers, in Figure 4). The highlight automatically moves to the next command when the current one is completed.

Most WIMP commands can be executed through several different methods. We are only interested in one method for each command, so the instruction window indicates the method with which each command should be executed. If a user selects the wrong command, an error is logged and the user must try again until the correct command is executed.

We created five document editing tasks involving only the 15 commands listed above. Each task contains a mixture of commands and methods that a user could reasonably use during document editing.

Task 1 – bold, italic, paste, insert table
Task 2 – font size, underline, save, print
Task 3 – insert page numbers, undo, apply style, center alignment
Task 4 – font size, toggle bullets, line spacing, italic
Task 5 – insert page numbers, undo, bold, copy

Grouping commands into tasks was done in order to present commands to users in reasonable chunks. The task groupings are not relevant in our hypotheses or analysis. The only goal was for the 5 tasks to collectively capture exactly 4 invocations of each of the five WIMP methods. (Five of the commands, one for each WIMP method, appear twice in the task set.)

6.2 Participants

Our study had 12 participants (3 female). Consistent with our focus on advanced computer users, all participants were graduate students: 3 in computer science, 6 in electrical and computer engineering, and 3 in mechanical engineering. All had significant Word 2003 experience. Participants received $30 for their time. To motivate quick and accurate performance, an additional $10 incentive was awarded to the top-third high performers.

6.3 Procedure

Each participant completed a single three-hour session. During the session, four distinct phases were completed:

Introduction – Participants were presented with a list of all the commands used in the experiment and asked to identify all of the ways that they knew how to execute each command in Microsoft Word. Next, they were given a demonstration of how GEKA works and then shown a printout containing all of the images used to represent commands in the study and the names of the commands and parameters that they represent. When participants felt comfortable with the images, the next phase began.

Performance testing – Participants completed a series of tasks in two separate conditions: one using WIMP methods only and one using GEKA only. Presentation order for the two conditions was counterbalanced.

Each condition began with a practice block, which consisted of each of the 15 commands being executed once. During the practice block, participants were able to look at the printout of command images and ask questions. After the first practice block, participants were given an overview of the rest of the study.

Each condition consisted of three blocks that were each made up of the five tasks repeated ten times each. The order of the five tasks was randomized for each participant and remained consistent across all blocks for each participant. Within a block, after each task was repeated ten times, participants took a break for at least 30 seconds, and after each block, participants took a break for at least 90 seconds. Participants were provided with magazines to peruse during the breaks.

Method choice - After the WIMP and GEKA conditions, participants were reminded of all the methods they knew for each command by going through their list from the introduction phase and executing each command once using each listed method and once using GEKA. Then, participants completed one final block using the same five tasks repeated three times each, in which they were able to choose any available method for each command.

Qualitative feedback – Finally, participants completed a questionnaire where they rated many aspects of their interactions with GEKA.

6.4 Dependent measures

Time was recorded for each command, from the moment the previous command was completed until the current command was correctly entered (there was an implicit error penalty). Time for each interaction method is the mean of the times for each of the four command executions using that method. Similarly, the number of errors made was recorded for each command. An error consisted of trying to execute an incorrect command, or a correct command with incorrect parameters. For an interaction method, errors are the sum of the errors for each of the four command executions using that method.

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 frequently changed methods during the repetitions. Finally, the questionnaire had participants rate the methods on a scale resembling the NASA TLX on 12 dimensions. The five WIMP methods were included as well as three cases for GEKA: zero-, one-, and multi-parameter commands.

6.5 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 a combination of explicit method choice and the qualitative questionnaire ratings.

6.6 Design

Because our hypotheses deal with the interaction methods rather than the task and command structure that participants saw, the data was collapsed into the following mixed factor design: 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, Greenhouse-Geisser corrections were used, which can be identified by non-integer *df*.

7 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.

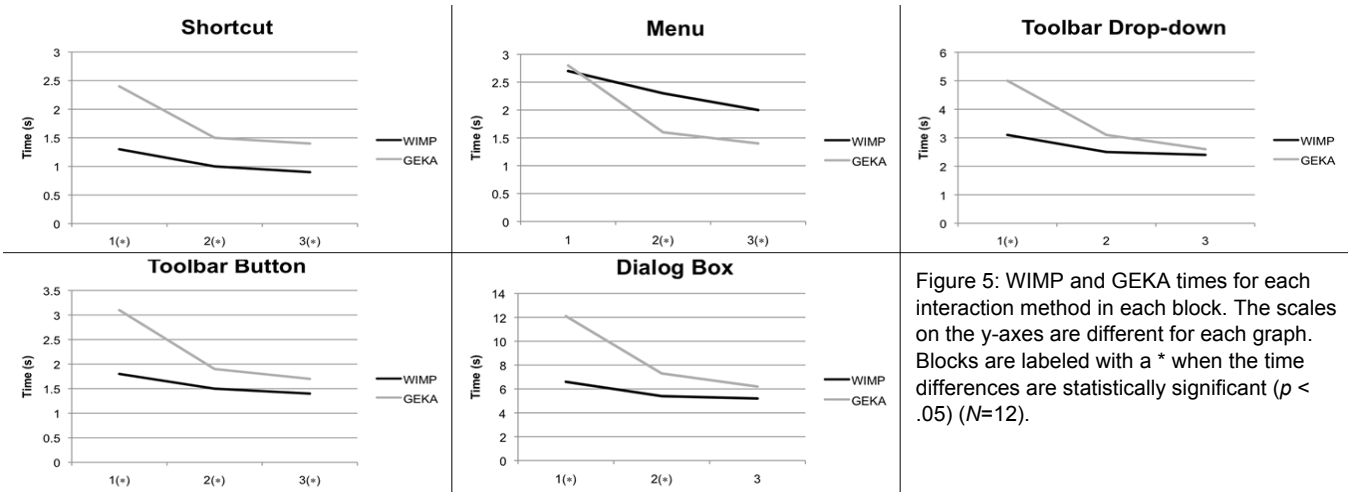

Figure 5: WIMP and GEKA times for each interaction method in each block. The scales on the y-axes are different for each graph. Blocks are labeled with a * when the time differences are statistically significant (p < .05) (N=12).

7.1 Time

GEKA is faster than menus, while shortcuts and toolbar buttons are faster than GEKA. The analysis is dominated by a 3-way interaction between condition, block, and interaction method ($F(2.65, 29.13) = 27.604$, $p < .001$, $\eta^2 = .715$). Figure 5 shows the breakdown. Pairwise comparisons show that in block 1 each WIMP method was significantly faster than its GEKA equivalent (all $p < .05$), except for menu, which showed no significant difference ($p = .817$). By block 3, WIMP was only significantly faster for shortcuts and toolbar buttons (both $p < .05$). (There was no significant difference for toolbar drop-downs, $p = .503$, or dialog boxes, $p = .102$). GEKA was significantly faster than menus ($p < .05$).

Performance is still improving with GEKA, less so for WIMP. Given the above 3-way interaction, we examined differences between blocks 2 and 3. In the WIMP condition there, was only a significant difference for toolbar buttons ($p = .042$) and borderline difference for menus ($p = .057$). For the GEKA equivalents, there were differences for toolbar buttons, drop-downs, menus, and dialog boxes (all $p < .05$).

Additionally, there were significant main effects of condition ($F(1, 11) = 13.320$, $p = .004$, $\eta^2 = .548$), block ($F(1.12, 12.35) = 115.610$, $p < .001$, $\eta^2 = .913$), 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 ($F(1.27, 13.96) = 19.670$, $p < .001$, $\eta^2 = .641$).

7.2 Errors

GEKA and WIMP have similar error rates. 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 1 shows the full breakdown of this interaction.

	WIMP	GEKA	sig.
shortcut	30	33	0.845
toolbar	15	23	0.136
drop-down	91	46	0.085
menu	43	31	0.394
dialog	75	123	0.071
total	254	256	0.948

Table 1: Total errors. Each method was used 1440 times per condition (N = 12).

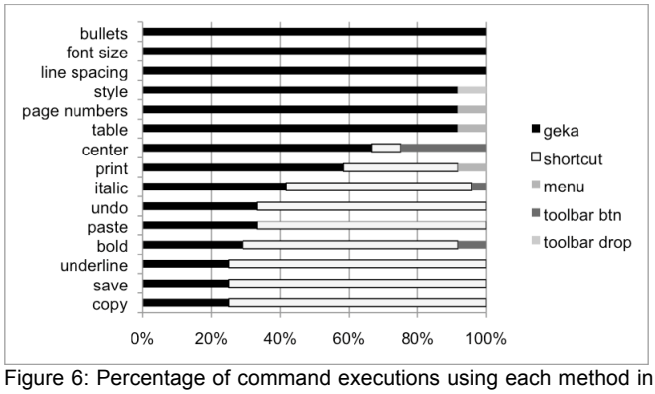

Figure 6: Percentage of command executions using each method in the method choice phase of the experiment (N=12).

7.3 Method Choice

GEKA is chosen overwhelmingly for commands with parameters. GEKA was chosen over toolbar drop-downs or dialog box. Figure 6 shows the breakdown.

Keyboard shortcuts are chosen for commands with no parameters, except for the commands center and toggle bullets. Participants generally did not know those shortcuts (one knew center and none knew toggle bullets). They chose to use GEKA rather than a mouse-based method.

7.4 Qualitative Findings

GEKA is rated significantly better than WIMP in all cases but shortcuts. Reliability analysis confirmed high consistency among the 12 dimensions (ease of learning, etc.) rated by participants (Cronbach's $\alpha = .966$), so we collapsed them into a single rating for brevity. Figure 7 shows the collapsed rating; a low score is better than a high one. A Friedman test on the transformed ratings showed a significant main effect of interaction method ($\chi2(7)=63.328$, $p < .001$) 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 ($p = .155$).

7.5 Summary

Hypotheses H1, H2, H4, and H5 were supported. H3 was partially supported, with no difference in speed, but a preference for GEKA. Overall, we found that GEKA's speed is competitive with WIMP. It is at least as fast as the mouse-based WIMP techniques, except for toolbar buttons, and is faster than menus. GEKA's error rates were comparable with WIMP, which shows that we have succeeded in creating a keyboard-based interaction method that supports recognition. Finally, GEKA is 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 mouse-based WIMP methods. We expect that other applications such as Quicksilver or Enso would perform similarly in a comparable experiment. However, we did note that most of our participants were using our built-in command abbreviations, which we expect would provide a noticeable advantage to GEKA.

8 Discussion

GEKA speed did not plateau, suggesting it could be faster. Performance did not plateau fully in either condition, despite repeating the same set of tasks many times. This points to the challenge of study design. We could not extend a single session beyond three hours; follow-on work will need to consider a multi-session study. The takeaway, however, is that speed of execution in GEKA was continuing to improve more so than in WIMP. This suggests that with further practice GEKA may outperform WIMP.

There are design opportunities to increase GEKA's speed. It is possible to reduce the number of keystrokes required to execute a command in GEKA either by using a single keystroke to enter GEKA mode or by using a quasi-mode like Enso does. 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.

Multiple-parameter commands are a key area for improvement. Our formative study showed dialog boxes to be slow and frustrating indicating an opportunity for GEKA to outperform against this method. Our current prototype does not yet achieve this goal. In our next design iteration of GEKA graphical feedback, we will focus on multiple-parameter commands to pursue this opportunity.

User's perceive GEKA to be faster than it actually is. In the method choice phase of the experiment, GEKA was consistently chosen over all mouse-based methods even though it was only actually faster than menus. This is consistent with responses on speed in the questionnaire. Those responses could have been biased by desire to please the researchers, but this is much less likely in the method choice because participants were aware that there was a monetary reward contingent on their performance.

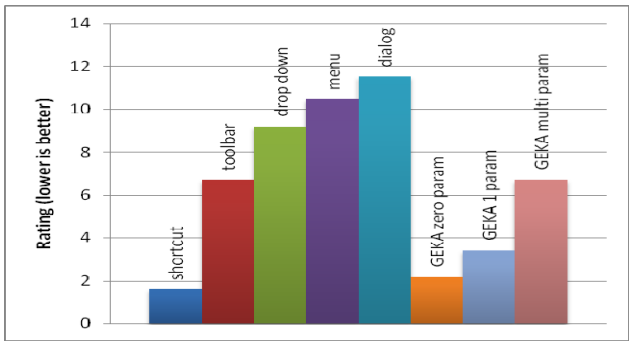

Figure 7: Collapsed questionnaire results (N = 12).

We have as yet no firm basis on which to conclude why GEKA feels faster than it truly is. We speculate that our participants found GEKA more pleasant to use than WIMP and therefore the time spent using GEKA seemed to pass more quickly. 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. Achieving pleasurable use is likely more important than a speed improvement alone.

9 Conclusions and Future Work

Our formative study suggested that advanced users are indeed frustrated with some aspects of WIMP interaction and desire alternatives. While we have not yet shown that GEKA is faster or less error-prone that WIMP in all cases, it is generally not any worse, and it was very well received in the method choice and qualitative feedback phases of our experiment. We believe that users' perception of a more pleasant interaction with an application is no less important than raw speed. Satisfied users are more likely to be productive and to continue to use an application.

Further study is required to understand GEKA interaction, including a longitudinal controlled study with a larger command set to examine users' performance over time, and a field study to examine how GEKA is used in actual work environments.

There are many potential benefits to the GEKA approach that we are excited to explore. These include advanced command line features such as scripting and piping, accessibility benefits for people who have physical difficulty using a pointing device, possible expansion of the GEKA technique beyond keyboards to support speech- or handwriting-based interfaces, and providing more flexibility in defining commands.

References

[1] Blacktree. (2009). Retrieved March 31. http://www.blacktree.com

[2] Gentner, D., and Nielson, J. (1996). The anti-Mac interface. In *Communications of the ACM 39*(8):70-82.

[3] Grossman, T., Dragicevic, P., and Balakrishnan, R. (2007). Strategies for accelerating on-line learning of hotkeys. In *Proc. CHI 2007*, 1591-1600.

[4] Humanized. (2009). Retrieved March 31. http://humanized.com

[5] Lane, D., Napier, H., Peres, C., and Sandor, A. (2005). The hidden costs of graphical user interfaces: The failure to make the transition from menus and icon tool bars to keyboard shortcuts. In *International Journal of Human-Computer Interaction*, 18:133-144.

[6] Linton, F., Joy, D., and Schaefer, H. (1999). Building user and expert models by long-term observation of application usage. In *Proc. Conference on User Modeling 1999*. 129-138.

[7] Microsoft. (2009). Search commands. Retrieved September 16. http://www.officelabs.com

[8] Miller, R. C., Chou, V. H., Bernstein, M., Little, G., Van Kleek, M., Karger, D., and schraefel, m. (2008). Inky: a sloppy command line for the web with rich visual feedback. In *Proc. UIST 2008*, 131-140.

[9] Mozilla Labs. (2009). Ubiquity. Retrieved December 12. https://mozillalabs.com/ubiquity/

[10] Norman, D. (2007). The next UI breakthrough: command lines. In *Interactions*, 14(3):44-45.

[11] Raskin, J. (2000). *The humane interface: New directions for designing interactive systems*. Addison-Wesley.

[12] Shneiderman, B. and Plaisant, C. 2004 *Designing the User Interface: Strategies for Effective Human-Computer Interaction (4th Edition)*.

[13] Stone, D., Jarrett, C., Woodroffe, M., Minoca, S. (2005). *User interface design and evaluation*.

[14] Wikipedia. (2009). Retrieved March 31. http://en.wikipedia.org/wiki/Command_line_completion