Dream Design of a Text Editor

Andrew Kowalczyk

December 06, 2013

1 Design

Text editors are some of the most important pieces of software that programmers can use for their craft. I think a redesign of this application using multiple interaction styles would be incredibly beneficial for a coder/programmer. For this particular assignment, I will model the way this application/package/plugin would work by emulating **Sublime Text 2** and **3**'s menus and actions. In this document, I will be using the Mac keys for shortcuts. ⌘ denotes Command. ⌥ denotes Option or Alt. ⇧ denotes Shift. ctrl denotes control.

The design would mainly use audio recording from the user for all main actions. When the user simply gets tired of typing, they can revert to their spoken voice for accomplishing their goal. The ability to type in the editor is always a possibility and can be relied upon in the event of the audio recording not working as planned.

Certain key combinations would denote certain types of actions. Whenever a user will hold the key combination, the application would wait for some kind of input and only after the user lets go of the keys will the application respond. This application would be advanced enough to be able to read and recognize variable and function names given by the user. The reason for choosing different key combinations for user input is because attempting to have the application attempt to decode what a user is trying to say is extremely hard. This way the application would know what kind of input it would be receiving.

When the user would launch the application for the first time, they application would ask them to read a passage. This is like the scene in *Mission: Impossible III* when Tom Cruise (disguised as Philip Seymour Hoffman) asks Philip Seymour Hoffman to read a passage [1]. This would analyze a person's accent and other vocal measurements so the application could truly understand their voice. They would then be guided through a series of tasks for them to be acquainted with the overall commands of the application. This would be comprehensive enough for them to be able to code from the get go. Also, this application could be modified to simply be an extension for any other text editor like *TextWrangler*, *Notepad++*, or *MacVim*. This way, cross application compatibility would be accomplished. For a list of all the commands, look at 14 for reference.

2 ⌨️File⌨️ actions

⎇ + Y Complete a specific file action. Input ranges from: "save", "save as", "save all", "new file", "close file", "close all files", "open most recent", "open last closed file", "close window", and "new window". Each respective action does what it denotes.

3 Edit

3.1 Edit actions

⎇ + E Completes a specific edit action. Input ranges from: "cut", "copy", "paste", and "paste and indent". Each respective action does what it denotes.

3.2 Undo and repeat

⎇ + Z Undoes or repeats an action. Input can be: "undo" or "repeat".

4 Selection

4.1 Selection actions

⎇ + S Selects a certain part of the text. Voice input can include: "all", "word", "line", "paragraph", "scope", "indentation", and "brackets". Any of the previous commands except for "all" can be prefixed with "end of" or "beginning of" to select from the cursor to the beginning or end of the word, line, paragraph, etc. Also, input can include: "right" and "left" to move the cursor one character over in either direction.

4.2 Selection manipulation

⎇ + S + M Manipulates the selection in several ways. Input includes: "duplicate", "comment", "uncomment", "indent", "insert line". "indent" can be suffixed with "left" or "right". "insert line" can be suffixed with "before" and "after". If the user has selected multiple subsequent lines, another input can be "join". This joins the selection on to one line.

4.3 Moving text

⎇ + M Moves the text. Input that the user can provide: "up", "down", "bottom", and "top". "up" and "down" can be suffixed with any number followed by "line" or "lines". For example, "up one line" or "down seventeen lines".

4.4 Case conversion and text styles

`⌥`+`T` Changes the style of a selection. Sample input includes: "upper case" and "lower case". If a user is preparing a document using LaTeX, some more input could be: "bold", "italic", "underlined", etc. This would allow them to change specific style of text in that document preparation system. Depending on what type of packages a user is using more commands could be defined for ease of use.

5 `Find`

5.1 Find

`⌥`+`F` Finds a word or phrase. Input is whatever the user wants to search for.

5.2 Find similar selections

`⌥`+`F`+`S` Finds next, previous, or all similar selections. Input ranges from: "next", "previous", "add next", and "all". This command would highlight the next, previous, or all instances of the given selection. It can also add the next instance.

5.3 Replace

`⌥`+`R` Replaces a selection. Input is whatever the user wants to replace the selection with.

5.4 Replace similar selections

`⌥`+`R`+`S` Replaces next, previous, or all similar selections. Input ranges from: "next", "previous", and "all". This command would replace the next, previous, or all instances of the given selection.

6 Text entry and manipulation

6.1 Text entry

1. `⌥`+`R` Records user input. This would allow the user to record any text they want to have entered. Keywords of whatever syntax the file is in would be recognized. Certain keywords would denote operators like $+$, $=$, $|$, $/$, etc.

2. `⌥`+`R`+`V` Records verbatim. This would tell the application to ignore keywords and to simply record whatever input the user has provided. This would allow for saying input like "for", "equals", "pipe", etc. so that the application would not be confused with what to enter into the file.

3. `⌥`+`D`+`V` Declares a variable. This command would take care of declaring a new variable. A user's preferences and the syntax of the file that the user is working on shape the way that variables would be declared. For example, a user could define the standard to be camel casing, or underscores between words, or hyphens. This preference could be assigned by language or just generally by the user. Depending on the language variable types definitely apply.

6.1.1 Deleting Text

`⌥`+`D` Delete a certain part of the text. This works extremely similarly to text selection. All of the same inputs work in the same way except for that the simply delete the text instead of selecting it. An added input here would be "delete" for the case that the user has selected some text and then decides to delete it after the selection was made.

7 Window/view manipulation

`⌥`+`W` Changes the view of the current file. Input includes: "up", "down", "page up", "page down" "top", and "bottom". This command would also allow for an input of "minimize" to minimize a window. User inputs of "left" and "right" would cycle through the active tabs of the window in the direction that was specified.

8 `Tools`

1. `⌥`+`B` Builds the file with whatever system is currently chosen. User input is: "build".

2. `⌥`+`B`+`S` Changes the build system to the users input. Examples include: "Python", "Make", "C++", "LaTeX", etc.

8.1 Preferences

`⌥`+`P` Changes a preference. Input for this can be: "font" and "color scheme". "font" can be suffixed with "larger" and "smaller". This command would not be able to edit and package control settings due to the way **Sublime Text** stores preference files.

9 `Other`

9.1 File syntax

`⌥`+`S`+`Y` Changes the syntax of the file. Sample input of the user includes: "Python", "C++", "LaTeX", "Ruby on Rails", "JSON", etc.

4

9.2   File encoding

⌥ + E Changes the encoding of the file and reopens it. Input from the user can be: "UTF-8", "UTF-16 Little Endian", etc.

9.3   Go to line number

⌥ + L Moves the cursor to a certain line number. Sample input can include a specific line number or the top or bottom. Examples: "fifty-three", "first", "last", "two", etc. These inputs can prefixed with "end of" and "beginning of" to go to the end or beginning of that line. The default will be the beginning of the line. This could be changed in the user preferences.

10   Error handling and response from the application

In terms of being able to undo any unwanted change made by the user, the command ⌘ + Z is always available to undo a task.

   In terms of error handling, if the application were to not understand what the user said, the application would prompt the user to re-speak their input after they let go of the key combination. If the input received was clearer than that, the application would do one of two things. First, if the input was extremely close to a specific keyword, variable name, etc. then it would prompt the user with, "did you mean XXX?" where XXX denotes the closest believe matching keyword, variable name, etc. If the input received was clearer yet still slightly indistinguishable between more than one keyword, the application would prompt the user a list of options that the application would believe that the user was saying. The user could then cycle through the options using "next" or "previous". If the application would prompt a user to correct something, the user would need to press ⌥ + H to respond to an error. This way the application would be able to provide positive,, specific, and constructive error messages for the user to correct their actions. In terms of providing constant feedback on the state of the application, the application would respond with "unable to complete" with a reason for not completing the action.

11   Usage Scenarios

11.1   Handicapped persons

The first user group that comes to mind are people who normally have some difficulty typing. Due to some physical barriers, they might be unable to input or manipulate text via the keyboard. This application would allow for the people who want to code to be able to do so.

11.2 The tired coder

The second user group that comes to mind is the coder who is simply just tired of typing. For this target group, the technology would simply be an addition to their flow of programming. For example, when a user wants to move a piece of code into another function, this application would perform the given task faster than the typing equivalent.

12 Rationale

12.1 Mental Models

The mental models of the designer (me) are to automate a task which is largely not automatic. Yes, there are many different applications that automatically produce code in varying ways from user input. Yet, automated code is almost always a big no-no. The mental model that this design is attempting to send across to the user is that this application/package/plugin is meant to work in conjunction with the normal commands that a user is used to using. This would not be a complete replacement of the way in which a coder could code, but more as a supplement.

12.2 Interaction design guidelines and principles

12.2.1 Guidelines

In terms of meeting certain guidelines regarding input and data entry, this proposed design is in line with Smith and Mosier's guidelines [4]. They mention that a design should strive for minimal input actions by the user. By using voice, the actual amount of key presses declines sharply. The interaction with the user is simplified incredibly and allows for less doing to accomplish the same result.

This also address the guideline that there should be a flexibility for user control of data entry. The voice input can always be used, but when a user wants to simply type out the code, they *can*.

12.3 Principles

A strong principle of interaction design is to know your user, which really means that you can't know your user. The design of any application is supposed to be able to work around many different dimensions. The most important dimension that the design of this text editor addresses is the dimension of physical abilities. By making this design work for people of any physical ability (provided they can speak), it covers a large group of people. By doing so, this application can accommodate multiple user profiles. This design also contains slightly multi layered interface due to the initial help task list for the user to get acquainted with the design.

Let's take a look and see how the design fares against some important rules of interaction design.

First, looking at a few "golden rules" of interaction design by Shneidermann, we can see that the design proposed takes a few into consideration. It caters to universal usability and it permits easy reversal of actions [3].

Second, looking at Nielsen's Ten Usability Heuristics, we can see that it address some on the nose. The design proposed uses simple and natural dialog and speaks the users language. These two are the case because the user is actually using their own voice for communication. It keeps things consistent by having the shortcuts always doing the same thing. It heavily employs the use of shortcuts for ease of use. And most importantly, it address the heuristic of help and documentation by having a help sequence always available to the user [2].

Third, looking at Tognazzini's Sixteen First Principles, it also makes sure that these are addressed. It protects the users work by always allowing them to undo what they have done. The interface provided is very explorable and the user would find many new commands at their leisure to employ. It employs the first principle of human interface objects by letting the user perceive the application visually. It plays to the user other sense because voice is a large component of this interface. It has a standard way of interacting and it has standard resulting behaviors. All in all, it is understandable, self-consistent, and stable [5].

All in all, the interface designed approaches many rules that are seen to be very important in the way that people interact with an interface.

13 Usability metric forecast

13.1 Learnability

The overall learnability of the application would not be the fastest. Learning a new set of key commands is never a quick thing to lean. Also learning new phrases for certain actions for users could take a while. Since the user would be guided through a series of tasks the very first time they opened the application, this would help with the learnability. This help sequence would be available for the user to reacquaint themselves at any time with the applications commands.

13.2 Efficiency

The efficiency of the application would increase over time. As a user became more and more familiar with the flow of the application, they would be able to use the application very efficiently. Since the list of actions is fairly comprehensive, the actions that a user could accomplish via voice is nearly the same as via keyboard.

13.3 Errors

The rate of errors would decrease over time. But as imagined, they could be pretty high for a first time user. In any case, the user would be able to simply ue the keyboard to accomplish their task instead of using the voice control.

13.4 Memorability

In terms of memorability, the application would fare well. The same commands would do that same things and revisiting this application after some time would not take long to remember how the application functions. Re-familiarizing oneself with the application would be straightforward. Once again, a user could look to the series of tasks outlined in the help section to reacquaint themselves with the application.
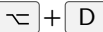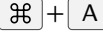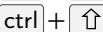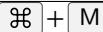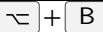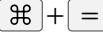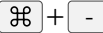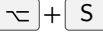
13.5 Satisfaction

Not surprisingly, this metric is largely subjective. For people who previously had a hard time coding due to a disability, this would induce large amounts of satisfaction. For people who do not have any physical disabilities impeding them from typing, their satisfaction would rely upon how efficient they have become at using the application. The overall learning process could be slightly hard, therefore unsatisfactory.

14   Key command table

If a voice input area is blank, this means that the user can input whatever they want. If the menu action and key commands are blank, this means there is no corresponding action or commands for the voice equivalent.

Key command	Voice input	Menu action	Key commands
⌥+Y	"save"	File ≫ Save	⌘+S
	"save as"	File ≫ Save As..	⇧+⌘+S
	"save all"	File ≫ Save All	⌥+⌘+S
	"new file"	File ≫ New File	⌘+N
	"close file"	File ≫ Close File	⌘+W
	"close all files"	File ≫ Close All Files	
	"open most recent"		
	"open last closed file"	File ≫ Open Recent ≫ Reopen Closed File	⇧+⌘+T
	"close window"	File ≫ Close Window	⇧+⌘+W
	"new window"	File ≫ New Window	⇧+⌘+N
⌥+E	"cut"	Edit ≫ Cut	⌘+X
	"copy"	Edit ≫ Copy	⌘+C
	"paste"	Edit ≫ Paste	⌘+V
	"paste and indent"	Edit ≫ Paste and Indent	⇧+⌘+V
⌥+Z	"undo"	Edit ≫ Undo	⌘+Z
	"repeat"	Edit ≫ Repeat	⌘+Y
⌥+S	"all"	Selection ≫ Select All	⌘+A
	"word"	Selection ≫ Expand Selection to Word	⌘+D
	"line"	Selection ≫ Expand Selection to Line	⌘+L
	"paragraph"	Selection ≫ Expand Selection to Paragraph	
	"scope"	Selection ≫ Expand Selection to Scope	⇧+⌘+▭
	"indentation"	Selection ≫ Expand Selection to Indentation	⇧+⌘+J
	"brackets"	Selection ≫ Expand Selection to Brackets	ctrl+⇧+M
⌥+S+M	"duplicate"	Selection ≫ Select All	⌘+A
	"comment", "uncomment"	Edit ≫ Comment ≫ Toggle Comment	⌘+/
	"indent right"	Edit ≫ Line ≫ Indent	⌘+]
	"indent left"	Edit ≫ Line ≫ Unindent	⌘+[
	"insert line"	Selection ≫ Expand Selection to Scope	⇧+⌘+▭
⌥+T	"upper case"	Edit ≫ Convert Case ≫ Upper Case	⌘+K, ⌘+U
	"lower case"	Edit ≫ Convert Case ≫ Lower Case	⌘+K, ⌘+L
⌥+F		Find ≫ Find...	⌘+F
⌥+F+S	"next"	Find ≫ Find Next	⌘+G
	"previous"	Find ≫ Find Previous	⇧+⌘+G
	"add next"	Find ≫ Quick Add Next	⌘+D
	"all"	Find ≫ Quick Find All	ctrl+⌘+G
⌥+R		Find ≫ Replace	⌥+⌘+F
	"previous"	Find ≫ Find Previous	⇧+⌘+G
	"add next"	Find ≫ Quick Add Next	⌘+D
	"all"	Find ≫ Quick Find All	ctrl+⌘+G
⌥+R+S	"next"	Find ≫ Replace Next	⌥+⌘+E
	"previous"		

Key command	Voice input	Menu action	Key commands
⌥ + D			
⌥ + D	"all" "word" "line" "paragraph" "scope" "indentation" "brackets"	Selection 》 Select All , Selection 》 Expand Selection to Word Edit 》 Text 》 Delete Line Selection 》 Expand Selection to Scope Selection 》 Expand Selection to Indentation Selection 》 Expand Selection to Brackets	⌘ + A , Delete ⌘ + D ctrl + ⇧ + K ⇧ + ⌘ + ⬚ ⇧ + ⌘ + J ctrl + ⇧ + M
⌥ + W	"minimize" "left" "rihgt"	Window 》 Minimize Goto 》 Switch File 》 Next File Goto 》 Switch File 》 Previous File	⌘ + M ⇧ + ⌘ + [or ⌥ + ⌘ + → ⇧ + ⌘ +] or ⌥ + ⌘ + ←
⌥ + B		Tools 》 Build	⌘ + B
⌥ + B + S		Tools 》 Build System	
⌥ + P	"font larger" "font smaller" "color scheme"	Sublime Text 2 》 Preferences 》 Font 》 Font Larger Sublime Text 2 》 Preferences 》 Font 》 Font Smaller Sublime Text 2 》 Preferences 》 Color Scheme	⌘ + = ⌘ + -
⌥ + S + Y		View 》 Syntax	
⌥ + E		File 》 Reopen With Encoding	
⌥ + L		Goto 》 Goto Line...	ctrl + G

References

[1] movieclips. Mission: Impossible 3 (5/8) movie clip - seeing boudle (2006) hd. `http://www.youtube.com/watch?v=CgX4uJSjOOY` (last accessed on 11/25/13).

[2] Jakob Nielsen. 10 usability heuristics for user interface design. `http://www.nngroup.com/articles/ten-usability-heuristics/` (last accessed on 11/24/13).

[3] Ben Shniederman. Shneiderman's "eight golden rules of interface design". `http://faculty.washington.edu/jtenenbg/courses/360/f04/sessions/schneidermanGoldenRules.html` (last accessed on 11/24/13).

[4] Sidney L. Smith and Jane N. Mosier. Guidelines for designing user interface software. `http://hcibib.org/sam/index.html` (last accessed on 11/24/13).

[5] Bruce Tognazzini. First principles of interaction design. `http://www.asktog.com/basics/firstPrinciples.html` (last accessed on 11/24/13).