

# 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.

The design would mainly use audio recording from the user for all main actions. The ability to type in the editor is always a possibility, but when the user simply gets tired of doing that they can revert to their spoken voice for accomplishing their goal.

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 would 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 text editor *TextWrangler*, *Notepad++*, or *MacVim*. This way, cross application compatibility would be functional.

## 2 File actions


⌘ + F 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

 + **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  $\text{\LaTeX}$ , 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 actions

#### 5.1.1 Finding similar selections



 +  Expands a selection of text in a number of ways. Input ranges from: "next", "previous", and "all". This command would highlight the next, previous, or all instances of the given selection.

## 6 Text entry and manipulation



### 6.1 Text entry

1.  +  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.  +  +  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.



#### 6.1.1 Deleting Text

 +  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 they 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.

### 6.2 Variable declaration

 +  Declares 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.

## 7 Window/view manipulation

 +  Changes the view of the current file. Input includes: "up", "down", "page up", "page down", "top", and "bottom". The speed at which the text is scrolled at could be

changed in the user preferences. 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++", "L<sup>A</sup>T<sub>E</sub>X", etc.

### 8.1 Preferences

⌘ + P The user would be able to scroll depending on their input: Sample input includes: "up", "down", "top", and "bottom". The speed at which the text is scrolled at could be changed in the settings.

### 8.2 User defined commands

## 9 Other

### 9.1 File syntax

⌘ + Y Changes the syntax of the file. Sample input of the user includes: "Python", "C++", "L<sup>A</sup>T<sub>E</sub>X", "Ruby on Rails", "JSON", etc.

### 9.2 File encoding

⌘ + E Changes the encoding of the file. 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 be 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 Usage Scenarios

### 10.1 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.

### 10.2 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. The action required for this is as follows: `⌘ + M` "move into function XXX". This involves less keystrokes and scrolling than the normal action: `⌘ + X`, then some scrolling, then `⌘ + V`.

## 11 Rationale

### 11.1 Priorities

### 11.2 Mental Models

### 11.3 Interaction design concepts, guidelines, principles, and theories

## 12 Usability metric forecast

### 12.1 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 learn. 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.

### 12.2 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.

### **12.3 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 use the keyboard to accomplish their task instead of using the voice control.



































### **12.4 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. Refamiliarizing 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.

### **12.5 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.

## 13 Key command table

Key command	Voice input	corresponding menu action
 + 		
ctrl +  + 		
 + 	"save" "save as"	 >   > 
 + 		
 +  + 		
 + 		
 + 		
 + 		
 +  + 		
 + 		
 + 		
 + 		
 + 		
		
 + 