Crimsas
A Text-Editor "Dream" Interface

Michael Fraser

December 6, 2013

Contents

1 Background

Common text editors have many of the same features, like language-specific keyword highlighting, automatic text indentation, and easy customization. These editors often use a system of menus and forms to allow the user to modify preferences, manage the file, or access additional tools. As with most applications, the text editors also allow the user to directly manipulate the shape of a session window, the location of the session window, and the positioning of session tabs. Most of the actual text writing is done by the typing of a keyboard, with no alternative forms of text input. Text editors are very commonly used by programmers, as they offer wide ranges of file types and programs do not require the unnecessary font editing capabilities of a word processor.

As all these text editors have many of the same features, they all maintain similar levels of efficiency, learnability, rememberability, errors, and satisfaction. The menus and forms that allow the users to easily find a desired command or tool allow for easy learnability of the editor. It also eliminates some need for high rememberability, as the forms and menus allow for commands to be quickly found again. The editors achieve similar levels of efficiency with the forms and menus, requiring several steps to reach each command. They also allow more experienced users to utilize increased efficiency with keyboard shortcuts. Satisfaction levels for the editors rely on the features of the editors and the ability for users to edit their preferences, and vary from editor to editor.

Many aspects and features of today's common text editors would be included in a the design of an ideal text editor, but current text editors lack several features that could bring text editing to a new level. As new technologies are developed, new methods of interaction are developed that could alter the way text editors allow for text to be input to a document. Text editors could even take some features of common word processors to make their use as programming tools more effective and satisfying.

2 Designing a New Interface

2.1 Aspects from Common Text Editors

2.1.1 Menus and Forms

While common text editors may not be the most ideal text editors, many features they have will be utilized in making a new interface. The menus and forms text editor's like Gedit utilize to organize commands and tools will be used in the ideal interface, as they allow for new users to easily search through all the available commands and tools. Keeping the forms and menus also allows new users to be more comfortable with the ideal text editor, as there will be a feature they recognize from other editors they may have used.

2.1.2 Keyboard Shortcuts

Keyboard shortcuts offer increased efficiency with utilizing tools and evoking commands, as they eliminate the need to search through the menus and forms to find the command or tool. Being able to quickly issue a command adds to user satisfaction as well, as the user can accomplish tasks without having to move their hands away from the keyboard. The keyboard shortcuts also provide consistency between different applications, as the many keyboard shortcuts are standard with many applications.

2.1.3 Window Sizing and Positioning on Click

As with most applications, mouse clicks and mouse drags allow the user to modify the size of an application window, the position of the window on screen, and the organization on tabs, if applicable. This direct manipulation aspect gives the users a sense of control, increasing their satisfaction. As this feature is standard with most applications, excluding it would limit how ideal the dream design can be.

Common symbols connected with these features include slashed lines on the corner of a window, to illustrate the window being dragged into a large window size, along with the cursor changing to a symbol indicating the edge of the window is modifiable.

2.1.4 Easily Variable Preferences

2.2 New Features

2.2.1 Voice Recognition and Input

Text editors limit the forms of input that can be used to modify the content of a document. They also limit the way commands and tools are selected or evoked. The ideal text editor would allow multiple forms of input and command selection, and natural language is an interface style that common text editors have not explored deeply. This feature would allow the user to produce documents without having to utilize their hands much at all. As many devices are already equipped with built-in microphones and secondary devices are readily available for other devices, natural language input is easily accessible.

This feature would never fully be deactivated unless the user has specifically disabled the feature, and their would instead be a form of "wait state" that the feature utilizes. During that "wait state," any voice input will be processed and words will be matched to libraries that can then be added to the document. This feature would eliminate some need of the keyboard, and provide a new form of content input that would make the application more flexible than other text editors. In a manner similar to the Xbox One Kinect system, where keywords alert the system to behave in a specific manner, this system would utilze a keyword that allows the user to say the name of a command or tool. The system would then evoke or select that command or tool for the user, with the user needing to search through menus and forms or utilize a keyboard shortcut. To ensure the keyword does not overlap with a potentially desired word input, the keyword needs to be unique to the application. For example, the Xbox One Kinect System waits for the word "Xbox" to be said, which is unique to the Xbox system.

2.2.2 Language Libraries and Autocorrect

A feature that this text editor will utilize is commonly found amongst word processors: autocorrect functionality. Autocorrect features note when the user has made a typing error and fix the error automatically. This is done by comparing the word with an error to the words of a word library. The word in the library with the closest resemblence to the mispelled word is then switched in to replace the mispelled word. In the event that the resemblence of any word in the library and the mispelled word does not reach a certain threshold comparison level, the autocorrect function instead highlights the word and offers suggestions to replace the mispelled word when the user investigates the highlight.

Autocorrecting is mostly used for natural language corrections, but libraries corresponding to programming languages can be utilized to allow this feature to apply to a wide variety of programming languages. This will allow users to build programs with greater accuracy the first time they write the program, eliminating some errors that would need to be cleaned through debugging.

2.2.3 Autocomplete

An autocomplete feature will add a new level of ease when it comes to adding content. As with bash commands in a terminal, the user will have the option to ask the program to complete the word that is currently being input (when the input is from the keyboard). The autocompletions will depend on the language being written, which will be autodetected by the application or manually set by the user. To utilize the autocomplete feature, and to stay consistent with other applications with the same feature, the "Tab" button will be utilized. Tab already has a special use in indentation though, so the autocomplete shortcut will be "Shift + Tab."

Adding an autocomplete function to the text editor will eliminate some need to remember all the vocabulary of a programming language, making it easier for a user to write code efficiently. The increased efficiency, along with lowering the need to consult documentation, will improve the satisfaction users have with the application.

2.3 Layout Features

2.3.1 The Toolbar

A search bar will be provided to allow users to search for a command or tool specific to the application, while allowing for the option to search a specific programming languages library for a keyword. This provides another way for the user to find a command, without them having to search through the menus and forms or remember the shortcut.

Any buttons in the toolbar will be given wide borders of operation, allowing the user to click anywhere near the option's text or icon to access the tool or command. This aspect apeases Fitz's law of user interaction, as the larger the button is the easier the button is to press, making the task of pressing a button more efficient. The button's will utilize icons that greatly correspond to the task or action they represent. The affordances of those icons will suggest to the user what it is that the button represents without the application needing to explicitly state the button's purpose. In the event the button's icon does not provide a clear picture of the tool it represents for the user, hovering a mouse over the icon will create a pop-up message with the command or tool's name.

A tab system will be included as one of the toolbars, allowing users to open multiple session windows from a single application window. The tabs are able to be reorganized by directly dragging the desired tab to a desired location. The tab feature and direct manipulation style of interacting with the tabs will not be new to text editors.

Scrolling through the document content of the new text editor will be similar to scrolling through

3 Design Analysis

3.1 Usability Metrics

The overall application will measure up against current text editor's when considering Nielsen's "usability metrics" of learnability, efficiency, errors, memorability, and satisfaction.

As the designed interface is a text editor, most users will already have some experience with similar applications and will be able to easily adapt to using the new interface. For the users with text editor experience, the application will be extremely learnable. Their previous experiences will allow them to understand the basics of a text editor so they can immediately begin learning the new features. The new features will be accompanied with tutorials, so that the user can easily step through the features. The tutorial will always be available also, so a user who has not used the application for some time can go back and look through the tutorials to figure out the various features. A user that is inexperienced with text editors will be able to use beginner tutorials that walk them through the purpose and uses of a text editor, the basic tools, and some best practices. When the user feels comfortable with the application, the more advanced tutorials will allow them to learn the more advanced features of the application. The tutorials increase how learnable the application is, and provide the users with a reference on how to use the application.

With multiple forms of input, multiple ways of evoking commands, and customizable user interfaces, the efficiency of the application relates to a user's preferred style of interaction. If they prefer the menus and forms to search for a command, then the user will be less efficient than an individual who chooses to utilize the shortcuts provided by the application. The overall efficiency would remain relatively high, as multiple options are available.

A user's ability to remember how to use the interface would be divided amongst the different features. Features common to text editors are often very basic, and a user would be very likely to remember how to utilize those features. Newer features, like the interface for natural language input, may require a quick glance at a help document or tutorial. This new feature is not typical to other applications, and the user would be less likely to remember the feature's uses after a significant period of time.

Keyboard text input is prone to typing errors, but a natural language input would be able to add content accurately if the user speaks concisely. It is less likely that typing mistakes or misclicks would occur when using natural language styled input, as those devices are not utilized in the process.

Satisfaction

3.2 Sytles of Interface