Death of the Design:

Have we forgotten what UI usability even is?

A survey of modern and old computer usability as composed by HackerSmacker

Control-X Control-F. myfile.txt. Control-E. Arrow around. This is a test. Meta-X. Control-X. Control-S.

A sight uncommon to the casual PC user, but, a site all too common to users of the Emacs text editor. Emacs is notorious for its immense power, amazing functionality, and horrendous user-interface design that was never subjected to usability tests; Emacs was designed at the hands of some hackers (old-days definition, before the 1995 movie of the same name came out to ruin the definition), not businesspeople that needed to use a computer on the regular. Is this a good user interface? Is this usable in the long run? What else can we glean from Emacs-style software when we pit it against software subjected to usability testing? What even is a good design? Is it easier to find good design than bad design?

The purpose of this essay is to explore several not-often-considered ideals in computer interface design. Tools are rarely designed with the intent of ergonomics being their top priority, but, there are a select few that are demonstrative of excellent "user longevity" -- a term I will explore in this essay that I am using to describe the inverse of user fatigue. I will also address these questions in a slightly different order than what I presented. Let us first address the final question: is it easier to identify a badly-designed computer user interface than a good one? My short answer is that it is, but my long answer is that we should not be so quick to make such hasty judgements. Emacs, mentioned before, was (in its antiquity) a common text editor on a myriad of computing platforms; born on the legendary PDP-10, Emacs provided a (comparatively) easy user interface around a very complicated text editor known as TECO. The Text Editor and COrrector, as it was known, is, bar none, a brilliant candidate for "worst designed program ever made." Somehow, it was made with the expectation that it would be used by all users of the computer (in this era, computing was "timesharing" -- more than one user used a computer at a time). This expectation falls flat on its face in the modern era; the commands for TECO look like someone randomly mashing keyboard keys. Without a reference manual, a new user would be completely and utterly lost. Emacs provided an (again, comparatively) easy to use interface wrapped around the monster that was TECO; its command-key driven and repetitive user-interface was seen as a miracle of computer usability compared to the

harsh world of TECO.

Fortunately, today, we no longer have to confuse ourselves with the difficult computing design of yesteryear... or so we think. Consider another hard-to-use program: vi. While a skilled user of the vi text editor on a UNIX (or its descendants and clones) system can edit files extremely quickly and find themselves very productive, much in the same way I do, the ergonomics of the program are *extremely poor*. vi runs in two modes: insert mode and command mode. In command mode, you can move about the document using the h, j, k, and l keys. Funnily enough, some users consider this to be very ergonomic, simply because you can use your four fingers without moving them (we will look at the implications of this later). You delete with x, start typing with a or i, delete a line with dd, so on and so forth. Simply put, the key choices of vi are intended to be mnemonic in nature; they are not intended to be ergonomic in the traditional sense, no, they are reminders of the function of the key. You can also press : to enter a variety of commands, or enter / to start searching. Now, contrast this with the UW PICO family of editors -- the nano editor is a common sight for new Linux users, and is a descendant of pico. The key combinations on nano are reminiscent of those of Emacs; control-key and meta-key ("meta" is the UNIX-world name for the "alt" key on a PC keyboard) key combos dominate the user input. What made vi special is that the save command, `:wq<enter>`, could be -- and often is -- entered using two different hands. This provides interesting speed implications, but fails to realize the true issue at play. With the Emacs and nano model, one is often restricted to entering most keybinds with their left hand only. Now, I concede that most PC keyboards (minus laptop keyboards) feature two control keys, left and right, but, this was not the case on the keyboards used in the era in which Emacs was designed. Keyboards then featured usually one control key, usually found to the left of where caps lock is found today on a PC keyboard. The result of this is that the user may hover their hand over the bottom left of the keyboard, ready to strike the control key and one or more keys at a moment's notice. As for vi, I found myself the victim of bizarre ergonomics: when I use vi, my hands do not rest on the home row. My left hand has my index finger on x, middle finger on w, and my ring finger on q. My right hand has a finger on i, colon, and enter. When I begin composing a document, much as I might on a typewriter on the days of old, my hands revert to proper keyboarding technique.

So, then, what gives? Is this a *bad* user interface? Yes. The user is slowed down by either the fact that they must reconfigure the muscle-memory of their normally-home-row typing scheme for the sole purpose of typing in vi, or be faced with perpetual slowness with no speedup in sight. Emacs and nano are no better -- it

is dominated by excessive dependance on the control key with no alleviation in sight for most one-control-key-having keyboard users (which, nowadays, consists of almost every laptop sold on the market). However, is this truly bad? No, it is a consequence of the design of the program and the hardware (in this paper, “the hardware” refers to the terminal, not the parts in the computer itself that dictate its computing performance) it is confined to. The design of the editors mentioned before was made on systems that lacked certain key elements that we have on our keyboards nowadays, the most notable of which is the distinct lack of function keys and certain modifier keys. The meta key (or alt, if you prefer), at least in the school of design that is a UNIX-style system, sends escape then the pressed key. For instance, if you wanted to type meta-x on Emacs to show the command field, you could press meta-x proper or press escape then x. Put simply, the UNIX model does not treat escape like its own key. But, I hear the reader saying, vi uses escape to get out of input mode! Yes, this is correct, and, this very realization is part of another point I raise about inconsistent UI. vi is a "modal editor" and reflects a world of computing we now no longer dominate ourselves with.

Modal editors were common in the early days of computing featuring CRT-based terminals. Before then, the common sight for an editor was a "line editor" -- examples include UNIX's ed, the CMS EDITor, and others. This was necessary as terminals then were printing terminals (sometimes called teletypes). The user would type in the command to go into input mode, enter whatever they wished to enter, then press the enter key (or enter the special sequence) to get out of input mode and return to command mode. Once in command mode, they could enter a variety of commands to manipulate the file or save/quit. Now, the clever reader will identify this sounding awfully similar to vi; indeed, it is. vi is an interesting stop in computing history, as it is an incomplete product of the transition period from typewriter-terminal-designed line editors to full-screen editors that would be common just a few years later. Some editors, like VM XEDIT, provided a similar interface (but took it further). XEDIT can be used both modally and interactively. One can enter an i on any of the command fields beside each line to insert, d to delete, so on and so forth (the user can often guess the command, funnily enough). They can then type in a blank line, or tab (note the appearance of keys foreign to UNIX users now) over to the command field, type save, or just press F3 to quit. Note the differing model here. This is part of the argument that I would like to make: some computer systems have no clear standard of usability, and don't fully use the hardware presented to them for maximum functionality.

Let us consider this last point, as I believe it is the most important part of my

discussion. I will assume the reader is familiar with the old ways of Windows: common user interface guidelines were seen as a semi-critical concern in software design. This was thanks in no small part to IBM, who developed something in the late 80s called Common User Access -- CUA. This was essentially a collection of common keybinds we now have come to know and love: control-C for copy, control-V to paste, control-S to save, F1 for help, et cetera. While we often take these keybinds for granted, consider the earlier examples of fragmented user interfaces within the UNIX (and friends) world. Under the CUA model, programs on DOS, Windows, OS/2, and mainframes (ISPF on MVS and VM/CMS fullscreen programs like OfficeVision) featured similar keybinds. Programs like DisplayWrite/370 would have had similar keybinds as DisplayWrite/2 on a PC; ISPF and the VM/CMS HELP feature used very similar keybinds (F3 to go back a screen, F12 to go to the main menu, F7 and F8 to scroll up and down, so on and so forth).

One of the consequences of a common user interface like CUA is that it can stifle interface flexibility. Consider vi: whereas control-C and control-V may be commonplace to PC software such as Word, vi achieves the same behavior with y and p -- y to yank, and p to put. What mnemonics exist for control-C and control-V? Sure, c is for copy, but, v has no clear meaning other than it is right next to the copy key. x is for cut, which makes some sense, but, this is where the user interface design models differ -- the vi software designers appear to have sought to place hotkeys based on mnemonic and ease-of-remembrance based on their function, but, the reality is more boring than that: vi is an extension of the ex editor, itself an extension of ed, and, the keybinds vi uses are merely mnemonics for commands that would have been seen on a line editor.

Even in the era of PCs, some programs featured vastly different user interface models even after CUA was introduced. Take, for instance, the program I used to compose this paper: WordPerfect. Granted, I am using the Windows version of WordPerfect, but, the DOS versions were notorious for their bizarre user interface design. Despite not following the unwritten standard of CUA, WordPerfect for DOS cut its own path with heavy use of function keys with modifiers. Many users came to love WordPerfect in the same way others came to love, for instance, Emacs or vi, but, few found a truly adequate migration path to other word processing software after WP's demise in the mid-90s. Despite there not being much of an option for users to turn to that were familiar with its keyboard user interface, some programs replicated the functionality of other aspects of its UI. This misplaced UI design mindset resulted in many differences cropping up throughout PC software in the 90s; the effects of this are still being felt today.

Consider, briefly, Microsoft Word: being a mainstay of personal computing, backwards compatibility is paramount. The latest version of Word can read documents from Word 6.0 (from 1994) with no trouble; most of the key combinations still apply to this very day. Often, people fail to identify the source of these key combinations: control-C to copy, control-V to paste, et cetera, are all IBM CUA combinations. CUA is certainly dominant in the world of commercial-software-powered PCs; the commercial UNIXes (for instance, IBM AIX) had management programs that featured user interface designs that followed other UI designs (for instance, smit on AIX is modelled after ISPF on MVS). Moreover, for a user of a non-US keyboard, the key combinations remain the same; no matter what language keyboard you are using, the key combinations remain in the same position. However, one must ask, what user-interface standards exist in the UNIX world?

Recall the user interface model of vi. Despite being rather un-ergonomic (and sometimes downright useless to users of non-US standard keyboards), it has cemented itself as a standard for UNIX programs to model themselves after. Most common are the h/j/k/l arrow keys; while this is not explicitly a problem if the user is running on a local terminal emulator session under an X desktop (for example), the user may incur massive issues attempting to use vi (or vim) over a telnet or ssh session without key position remapping. This results in a user that is using a non-US keyboard layout heavily confused attempting to use vi; something that could be easily alleviated by using the real arrow keys on the keyboard, heavy use of function keys, and using Curses's raw mode.

Another point I wish to consider is how a user might learn to use these programs back in their former years -- how would a user learn to use vi? If the user comes from the PC world, they may press F1 for help. Unfortunately, F1 is about as useless as throwing a paper airplane at the computer when it comes to vi, and, typing `help vi` from the command line does not help the user any either (though it will encourage them to read the approximately-as-useless UNIX manual page on vi). At least with office software (word processors like WordPerfect, Microsoft Word, Lotus WordPro, etc) of the past, they do possess an option to have the user run through a tutorial. Where is the vi tutorial? Yes, vim has a tutorial, but, many users even find that complicated and defer the tutorial for a YouTube video.

In conclusion, I believe we, as programmers, should fully use the best tools we have for constructing user interfaces, and stick to a standard; to not stick to a standard is, in my eyes, against the user. While yes, the “user” of a modern Linux system is more technical (most likely) than the average office-dwelling Windows user, there

is no excuse for over-complicated UI design as judged by end users. Overly complex interfaces and programs scare off the “end user” and don’t encourage them to branch out and try new systems. However, the reverse is also true and can present a “skill ceiling:” a skilled operator can fly through Windows with only a keyboard thanks to all the modifier key combos (alt-space, alt-tab, arrows, enter, space, alt-escape, etc). Furthermore, we should not stick keys to mnemonics -- `:wq` may not line up at all with the expected "mnemonic keys" intended to recall or abbreviate a function on a non-QWERTY keyboard layout. Instead, we should spread the load out; use the function key-oriented and modifier key-combo methods equally. We should also place a strong emphasis on usability testing; without adequate usability testing, how can we judge a user’s productivity in a program? Should we bend over for the average user entirely? Has the “vi model” ran its course? Is this all a matter of personal preference, or is there an actual objective science to this?

In followup papers, I wish to discuss these questions.

This paper was made possible with contributions from users with the following internet aliases: notatypewriter, averageemogirl, minneelyyyy, and possibly others.