New Command Line Interfaces

Nilanthi Seneviratne
Department of Software Engineering
University of Auckland
lsen008@ec.auckland.ac.nz

ABSTRACT
Most interfaces currently use graphical metaphors to show their functionality. However encapsulating an action within an icon is difficult. In most cases software developers will insert tooltips to help the user understand the actions. In this case a command line interface can be very effective because it can give a concise explanation of an action to the user via text. However it is generally considered that command lines have a very steep learning curve because users need to learn a set amount of commands in order to operate the interface effectively. This paper will be discussing the benefits of using the command line, issues related to the command line and how new command line interfaces attempt to reduce these issues.

The cognitive load on the user can be addressed via a command line interface which would reduce the number of commands the user needs to know or explicitly let the user know which commands are available at a given time. Also the command line should not require the user to learn all commands or one which would degrade gracefully to return information directly relevant to the command entered. Such command line interfaces exist today in search engines such as Google, Yahoo, and Live.com.

Author Keywords
command line interfaces, visual, search engines, menu based command line.

INTODUCTION
Command line interface (CLI) requires the user to interact with the computer via text whereas the graphical user interface (GUI) allows the user to interact with the computer using windows, icons, menus and a pointing device.

The CLI is a powerful interface which allows the user to create shortcut commands via piping or using flags. For example in a UNIX shell a user can search for a certain pattern of characters (word) in different ASCII text files by performing the "grep" command. If the user needs to know how many instances this same pattern occurs in all of the files he or she could send (pipe) the output to the command which counts words (wc). However in the GUI the user would have to open all the text files and perform an in-file search for the word on each of the files. It would also certainly be impossible to count the number of times the word would be occurring in total. Yet, this interface is considered to be less usable by HCI experts (Thompson et al., 2007). This paper will be discussing as to why this may be and also how new interfaces try to redeem this image of the CLI.

BENEFITS OF COMMAND LINE INTERFACES
There are many benefits gained by using command lines interfaces (CLI). Mainly time taken to complete a task can be reduced by creating complex commands by stringing simple commands together (Thompson et al., 2007) (Westerman, 1997). This gives the user fine grained control when using the computer (Thompson et al., 2007). It also gives developers a less confusing way to describe actions.

Making sense of commands
The main advantage of using a CLI is that the user can build powerful commands by ordering more simple commands together (Bland et al., 2007) or by activating flags. For example in a UNIX shell a user can create a hierarchy of directories by typing "mkdir –p /test/folder/structure". If the user were to create this same folder structure in a GUI the he or she would have to create each folder one at a time. This sort of flexibility allows the user to create custom commands (or scripts) to automate processing of data (Bland et al., 2007) (Thompson et al., 2007) and retain fine grained control over the computer.

Developers can possibly reduce the cognitive load exerted on users by using meaningful commands to portray actions available to the user. This is because graphical user interfaces (GUI) use metaphors in the form of icons to show functionality of software. However this would work well if we are to portray non-abstract concepts such as physical objects. If we are to explain abstract concepts, words can be used more effectively than icons (Raskin, 2008). CLI uses words to describe actions the computer can take. Thus it can be better equipped to give concise explanation of abstract actions such as "Save", "Import" etc. than icons. A study conducted by Durham and Emurian (1998) also state that novice users consider CLI less confusing than menu driven interfaces.

ISSUES RELATED TO COMMAND LINES

The main reason that the command line interface (CLI) is regarded as user unfriendly is because it requires the user to recall all commands prior to the use of the interface (Raskin, 2008). Due to this there is a high error rate as well as a higher learning curve when using the CLI (Bland et al., 2007) (Durham and Emurian, 1998). There are other issues such as the difficulty to distinguish patterns and ineffectiveness of displaying how a system works (Thompson et al., 2007) (Morgan et al., 1991).

Command Related Issues

All command line interfaces do not display all their commands to the user. Rather it requires the user to recall commands in order to interact with the interface. This can be intimidating to a user as well as time consuming, especially when they do not know which command to use. Thus it is expected that expert computer users would use the command line more often than a novice computer user.

Westerman (1997) conducted a study comparing the cognitive load imposed on the user by graphical user interface (GUI) and CLI in relation to the expertise and the cognitive ability of the user. This study found expert users with high associative memory tend to use the command line more than novice users. This finding is also supported by Durham and Emurian (1998). Due to this the novice user cannot take advantage of the powerful command processing of the CLI and finish their task in less time.

Failure to remember commands is further accentuated by having less memorable command names (Raskin, 2008). For example in order to unzip a tar file the user should type "tar –xfvz path/to/tarfile". Here the user has to remember two aspects of the command: the actual command name and all flags associated with the command. If the user types just the command and give the tar file path the command would not unzip the file, rather it requires the user to type the appropriate flags to unzip a tar file (in this case it is xfvz).

The CLI also requires the user to type in the commands in a strict syntax (Westerman, 1997). If the user is unable to remember the syntax or does not know the syntax they would need to spend time in finding the correct syntax (Thompson et al, 2007). For example if the user needs to find more information on the "tar" command he or she could type "help tar" but this syntax is not supported via the command line (see Figure 1). Thus it generates a generic error message. This process itself would delay the user from completing the intended task. If however the user was using a GUI then the command for unzipping a file can be selected from a drop down menu. It was also noted by Morgan et al. (Morgan et al., 1991) that "Help" command is typed more often in the CLI rather than the GUI.

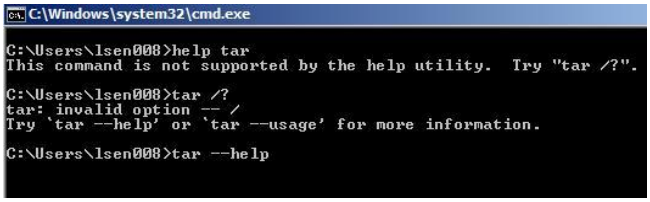

Figure 1: The user needs to type three times to get the right syntax for help

Mental Mapping of the System and Pattern Recognition

Morgan et al. (1991) conducted a study which aimed to compare and contrast the cognitive load imposed by the graphical user interface (GUI) and the command line interface (CLI). Their study found that the number of syntax and semantic errors generated when using the CLI was almost even. However when using the GUI the syntax errors were greater than the semantic errors (see Figure 2). From this they concluded that when using the GUI, the interface asserts a certain mental map of how the system works. For example in a GUI the contents of a folder is displayed to the user, whereas in a CLI the user has to query for the contents of the folder. This reduces the cognitive load on the user, thus the number of semantic errors were reduced.

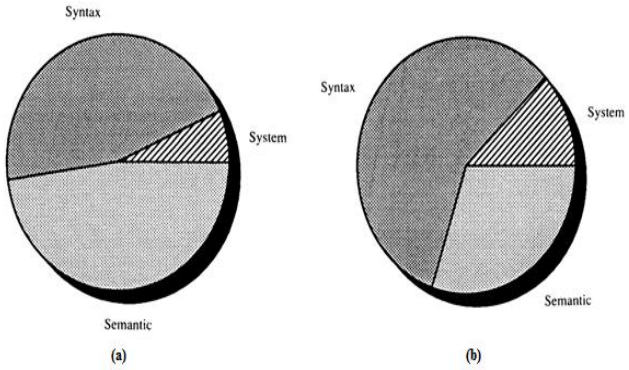

Figure 2: Syntax errors and semantic errors when using a CLI (a) and GUI (b) (Morgan et al., 1991)

Thompson et al. (2007) also conducted a survey comparing the GUI with the CLI in a network intrusion detection environment. In order to detect an intrusion in to the network, the security engineer would match the data presented by the network with patterns of intrusion attacks. If there is a match between the data and the data pattern then the engineer would be able respond to that attack. Here they found that CLIs were not able to display information on intrusion patterns. Their test subjects commented that if the values are updated frequently it is hard to keep track of them and thus harder to detect patterns. Thus the study proposes that there should be better pattern matching commands in the CLI. Although day-to-day users of computers would not be involved in intrusion detection, it is important to note that specialized users of CLI should be able to use the interface with ease.

COMMAND LINE INTERFACES FOR EVERYONE

Most new command line interfaces try to lessen the cognitive load on the user by having memorable command names or allowing them to visualize the commands. Another option would be to provide a menu of the commands rather than expecting the user to remember all the commands. Auto-complete, context sensitive help, providing the user with possible commands while typing and allowing the user to rename confusing commands can also help decrease the cognitive load on the user.

There are several CLIs which implement these solutions. Search engines, linguistic command lines, menu based command lines, command lines for routers and visual command lines are some examples of such interfaces.

Visual Command Line Interface

Glaser et al. (1995) has developed a visual command line interface (CLI) named P-sh, aimed at Mac OS users. It allows the user to write scripts as well as perform commands using one interface. This interface supplies the user with a programming interface which they can modify, create and test scripts without changing applications. This shell is developed using Prograph to resemble a UNIX shell with the functionality of AppleScript. This interface allows the user to add commands using visual metaphors (e.g. boxes for methods, lines for showing data flow etc.) to create complex scripts and/or complex commands which can be executable by the CLI.

Menu Based Command Line Interfaces

Bland et al. (2007) has developed a menu based CLI which merges the menus of GUI with the command line interface. This interface was specifically developed for Open Source Cluster Application Resources (OSCAR) toolkit. One of the main motivations for this project was to allow the automation of installing and testing OSCAR. Another reason for implementing a CLI was to allow advanced users and/or developers to have more fine grained control over the installation and maintenance process.

Answer Engines not Search Engines

In the opinion of Norman (2007) search engines have been transformed into answer engines via a relaxed form of the command line interface (CLI). According to Norman people are typing commands rather than keywords into the search engines. Google allows users to type "define:" command with a keyword and it will only gather web pages from popular online dictionaries list them in the order of relevance Norman also suggests that this search line interaction can further be strengthened by allowing the user to tag files and mail with keywords. For example rather than having a file structure to store emails Gmail allows the user to tag their emails, and search these emails via tags. This means that different emails can be collected together by common tags.

The Linguistic Command Line

Currently there are only two linguistic command line implementations available (Humanized's Enso and Blacktree's QuickSilver). The aim of Enso (Raskin, 2008) is to provide global functionality independent of programs which it is operated on. For example a user can select some text type "spell-check" on Enso to perform a spell-check. This spell check is independent of any program which is currently open. It uses more 'natural' language syntax so the user does not have to remember strange command names. Natural language is not fully supported by this command line thus it depends on suggesting commands to the user and auto completing the commands.

JUNOS and CAD Command Line Interface

JUNOS (Gredler and Goralski, 2005) is a command line interface developed especially for routers. It was developed in 1986 by Cisco Systems Ltd. It implements an ASCII based interface which writes information about the operation of the router into human readable text files. It uses simple commands which are similar to UNIX commands.

The computer assisted design (CAD) command line interface was used in an evaluation of user interfaces in a CAD environment conducted by Roy (1992). It was the preliminary interface used for modeling objects using the computer.

Easing the Burden on the User

All command line interfaces described above aim to lessen the cognitive load on the user. However they implement different methodologies to achieve this.

Usually if the user needs to perform a command she or he would type the command into the command line and press "Enter". If they wanted to script several commands together then they will open a text editor and write the commands and save the file. In order to execute this script they will have to either double click on this file or invoke it via the command line. Using P-sh the user is able to write sequences of commands as well as execute individual commands within one application. This reduces the cognitive load on the user because they would only have to learn one set of symbols (because the application is a visual representation of the command line), for all command line and scripting tasks. This interface however still requires the user to know which commands perform which action and the syntax of the commands.

In contrast answer engines do not require the user to know the syntax of the commands. They also tend to implement a more 'natural' language to describe commands. This is mainly because most if not all users of answer engines, do not use the command line and/or would not understand complex and confusing command names. Another advantage answer engines have over P-sh or any other command line interface is that it is able to degrade gracefully into a search engine if the user mistypes the command or does not use the correct syntax. They also may implement a spellchecker so that if the command is mistyped then it would give the closest matching command as an alternative (e.g. Google's 'Did you mean…"

functionality). This reduces the cognitive load on the user because they are not required to know any of the commands or the syntax of the commands.

The linguistic command line (Enso) aims to reduce the cognitive load by employing auto completion of commands and suggesting possible commands while the user types into the interface. It is similar to search engines because it provides support for 'natural' language syntax although not as much as the search engine. In quasi-modal mode the user is required to hold down the activation key throughout typing of the command. The command is only executed once the user releases the activation key. This may make the application cumbersome to use (especially since the activation key is defaulted to "Caps Lock" key) however the application provides a 'sticky' mode where the user is required to only press the activation key once and the command is executed once the user presses enter.

The computer assisted design (CAD) command line reduces the cognitive load on the user in a similar way to Enso and search engines, by supporting limited 'natural' language syntax. However it also allows the user to rename command names to suit their own. This functionality can be helpful because the user can change any confusing or long command name into simple and short commands.

In menu based command lines the user is directed as to which commands are available to them at all times. This means that not all commands are available at all times especially if the command line follows a state-machine approach. Although this reduces the complexity of using the command line it doesn't allow for flexible interaction (in terms of allowing the user to do whatever they wish at anytime) with the command line.

JUNOS is has similar attributes to a UNIX type command line interface. Thus it is privy to some short comings of the UNIX command line. It has similar commands (which can sometimes be hard to remember) but because of this any user who has used the UNIX shell can operate the interface easily. It also uses short cut keys of EMACS (a text editor heavily used by the UNIX community) to navigate the cursor. This again would be useful for users who have experience in using EMACS. Thus for the most time JUNOS relies on the users being familiar with an UNIX environment.

However it does provide some functions as to reduce the cognitive load on users who are not familiar with the UNIX environment. This includes contextual help and auto complete. The auto complete function is similar to auto complete functions provided by most command line interfaces. This means it requires the user to type in the start of the command and press "Tab" key. If the command is correct it will complete it, however if there are several commands starting with the same characters it will list all of the commands. This auto complete does not provide the arguments needed for the command. It is required by the user to remember which arguments are needed for each command. Context sensitive help conversely presents only relevant commands depending on other arguments provided.

The command line parser also accepts commands that are half typed. This however is done only when the parser determines that the command is unambiguous. For example instead of typing "show isis database" the user can type "sh is d". This reduces the amount of commands the user needs to know and also auto complete and context sensitive help allows the user to not know all the commands used in the interface.

Although the above command lines have reduced the cognitive load on the user, it seems that from studies conducted (Durham and Emurian, 1998) (Westerman, 1997) (Morgan et al., 1991), novice computer users prefer the GUI. This can be because it is easier for the user to look at the GUI and find a command for their task, especially when they do not know how to approach a task. However in a CLI the user is not given any clues as to how to approach the task. For example the user needs to guess what command they think, they would need to complete their task. It seems that CLI performs better if the user has some idea of how to approach their task.

CONCLUSION

The command line interface can reduce the time taken by allowing the user to create powerful commands to analyze data and operate the computer. It also offers a concise way of explaining functionality of software to the user.

However current implementations of command line interface (CLI) use commands that are hard to use, and have a rigid syntax for entering the commands. Some do not intuitively display any information to the user on how the system has been constructed.

In order to improve the CLI, developers should consider implementing memorable command names, limit the number of commands available to the user or support at least a limited form of 'natural' language syntax. New implementations of CLI should hint the user about possible commands via auto complete or suggest matching commands.

There are several CLIs which implement these functions. These include the linguistic command line, P-sh, modern search engines, menu based CLI, JUNOS and CAD CLI. However it seems that these command lines would be helpful for normal users only if they have some idea of how to approach their task.

FUTURE WORK

Although new command line interfaces (CLI) reduce the cognitive load, there has been little work done on generating a clear mental map of the system. For example if the user changes the directory which they currently in the interface could automatically list the files in the new directory. Also there can be different colours used to denote file and sub folders. Displaying patterns of information is

handled poorly in CLI. Fast changing values are not highlighted and displaying of data is not intuitive (i.e. Listing information one after another in one column. This may not be usable if the user has to scroll up and down the interface to compare results or two values.).

REFERENCES

Bland, W., Naughton, T., Vallee, G. & Scott, S. L. (2007) *Design and Implementation of a Menu Based OSCAR Command Line Interface*. Paper presented at High Performance Computing Systems and Applications. Retrieved March, 22, 2008 from http://ieeexplore.ieee.org.ezproxy.auckland.ac.nz/iel5/4215545/4215546/04215574.pdf?tp=&arnumber=4215574&isnumber=4215546

Durham, A. G. & Emurian, H. H. (1998). Learning and Retention with a Menu and a Command Line Interface. *Computers in Human Behavior*, *14*(4), 591-620. Retrieved April, 19, 2008 from http://www.sciencedirect.com.ezproxy.auckland.ac.nz/science?_ob=MImg&_imagekey=B6VDC-3VF9DYS-5-H&_cdi=5979&_user=140507&_orig=search&_coverDate=12%2F01%2F1998&_sk=999859995&view=c&wchp=dGLbVlz-zSkWA&md5=e02d26e36827318bc96013ad1e201a4d&ie=/sdarticle.pdf

Glaser, H. & Smedley T. J. (1995). *PSH-The Next Generation of Command Line Interfaces*. Paper presented at 11th International IEEE Symposium on Visual Languages. Retrieved March, 24, 2008 from http://portal.acm.org.ezproxy.auckland.ac.nz/ft_gateway.cfm?id=834288&type=external&coll=GUIDE&dl=GUIDE&CFID=22225474&CFTOKEN=68917359

Gredler, H. & Goralski W. (2004). *The Complete IS-IS Routing Protocol*. Kent, United Kingdom. SpringerLink. Retrieved April, 19, 2008 from http://www.springerlink.com.ezproxy.auckland.ac.nz/content/w6516p5247hk6747/fulltext.pdf

Morgan, K., Morris R. L. & Gibbs, S. (1991). When does a Mouse become a Rat? or… Comparing Performance and Preferences in Direct Manipulation and Command Line Environment. *The Computer Journal*, *34*(3), 265-271. Retrieved April, 19, 2008 from http://comjnl.oxfordjournals.org.ezproxy.auckland.ac.nz/cgi/content/abstract/34/3/265

Norman, D. (2007). The Next UI Breakthrough: Command Lines. *Interactions*, *14*(3), 44-45. Retrieved March, 22, 2008, from http://portal.acm.org.ezproxy.auckland.ac.nz/ft_gateway.cfm?id=1242449&type=pdf&coll=ACM&dl=ACM&CFID=22221489&CFTOKEN=19136043

Raskin, A. (2008). The Linguistic Command Line. *Interactions*, *15*(1), 19-22. Retrieved March, 21, 2008, from http://portal.acm.org.ezproxy.auckland.ac.nz/ft_gateway.cfm?id=1330535&type=pdf&coll=ACM&dl=ACM&CFID=22221489&CFTOKEN=19136043

Roy, G. G. (1992). An Evaluation of Command line and Menu Interfaces in a CAD Environment. *International Journal of Computer Integrated Manufacturing*, *5*(2), 94-106. Retrieved April, 23, 2008 from http://www.informaworld.com.ezproxy.auckland.ac.nz/smpp/ftinterface~content=a777708821~fulltext=713240930

Thompson, R. S., Rantanen, E. M., Yurcik, W. & Bailey, B. P. (2007). *Command Line or Pretty Lines? Comparing Textual and Visual Interfaces for Intrusion Detection*. Paper presented at Conference on Human Factors in Computing Systems. Retrieved April, 23, 2008 from http://delivery.acm.org.ezproxy.auckland.ac.nz/10.1145/1250000/1240807/p1205-thompson.pdf?key1=1240807&key2=7341298021&coll=ACM&dl=ACM&CFID=65029996&CFTOKEN=43372283

Westerman, S. J. (1997). Individual Differences in the Use of Command Line and Menu Computer Interfaces. *International Journal of Human-Computer Interaction*, *9*(2), 183 - 198. Retrieved April, 19, 2008 from http://web.ebscohost.com.ezproxy.auckland.ac.nz/ehost/pdf?vid=2&hid=104&sid=a4cbb44e-967d-430b-b965-597e774a4240%40sessionmgr108