https://doi.org/10.48009/2_iis_2008_551-559

X-WINDOWS, GUI PROGRAMMING, AND MICROSOFT WINDOWS

Herbert Carew, University of Houston-Victoria, wescarew@gmail.com
Meledath Damodaran, University of Houston-Victoria, damodaranm@uhv.edu

ABSTRACT

Every operating system has an underlying graphical user-interface that it supports. The graphical user-interface under UNIX is X-Windows. X-Windows has a rich and historical origin with capabilities that other GUI packages do not support. X-Windows is an important graphic tool; other tools exists that compliment X-Windows based development in one way or another. The technology behind X-Windows spawned a generation of similar types of graphical user interfaces on disparate operating systems; hence X-Windows can be called the father of all graphical user interfaces. X-Windows provides an excellent foundation for user-interface development in a UNIX environment, yet supports usage with Microsoft Windows. In this paper we first provide a glimpse to the rich history behind X-Windows. We also examine programming considerations in X-Windows based development, where we also compare some of the methods within the X-Windows environment versus the Microsoft windows environment. Each window system has its own place, but there may be more to X-Windows than what many students and graduates of a typical IS program are exposed to. We will present a view as to why a software architect or engineer should approach a solution requiring a graphical interface with an unbiased focus and consider all tools and languages available, including libraries, network capabilities, spawning of images across networks to other computers, database considerations, debuggers, and source code maintenance.

Keywords: Graphical user-interface programming, X-Windows, Microsoft Windows, Application interface presentation.

INTRODUCTION

This paper will first present the history and background of X-Windows as a foundation for user interface design in software development. X-Windows' origin is addressed to reflect its influence and connection with other associated technologies. The X-Window based user-interface solution of today is the culmination of years of software and hardware evolution. The foundation presented in this study is based on projects involving software and hardware dating from 1972 through today.

We will examine programming considerations in X-Windows based development. Here we compare the methods within the X-Windows environment versus the Microsoft windows environment. Comparison of advantages and capabilities within the X-Windows environment versus the Microsoft windows environment will be discussed. Each window system has its own place, but there may be more to X-Windows than what a typical student or graduate of an IS program is aware of.

Available software and hardware tools are the most important building blocks of a project. A view will be presented as to why a software architect or engineer should approach a solution requiring a graphics interface with an unbiased focus and consider all tools and languages available. This will include libraries, network capabilities, spawning of images across networks to other computers, database considerations, debuggers, and source code maintenance.

X-Windows is not the means to a solution but the beginning. We mention this to remind ourselves of the many times we have seen programmers focused on a GUI to such an extent that they do not consider other software tools that could have aided in the solution to the application. X-Windows is no different than Microsoft Windows in that it is just a GUI and just an interface to the user. If the application fails, so does the GUI. The magic is when a programmer understands what the correct ingredients are to make the whole thing sparkle.

Software and computer hardware are just tools whether associated in a UNIX or Microsoft or any other environment. The right combination of software is needed for an efficient solution to occur, and the combination will differ from case to case. This can be compared to a carpenter with a bag of tools. If a screw is required to be inserted into a piece of wood, the correct tool must be pulled from the bag. Should I pull out a sledge hammer or a pair of pliers? Now, if the carpenter had a mentor to work under or went to courses on carpentry, the carpenter may have learned to pull out the screw driver as the

correct tool. Software engineering is no different. A software engineer should consider all tools available when given a project. An experienced mentor is part of that tool set. A mentor can pass on experience, shorten a project, and contribute to a better product at the end.

The software tool bag may contain a mix of UNIX, Linux, and Microsoft Windows operating systems. The language suite available might be C/C++, assembly language, Java, ALGOL, Visual Basic, FORTRAN, or COBOL. This is a wide spread of tools and most are disparate in nature. Today, new software engineers may be biased and may encourage the use of only one type of operating system and only one type of language for the project. The solution may fail, or perform sub-optimally, because of the selection and lack of fit of the software architecture with the requirements of the project. An example could be the use of Microsoft Windows in a real-time application environment such as the Space Shuttle used by NASA. Half way through the trip through space, a blue screen of death appears, and one has to reboot with <cntrl> <alt> <del>. This may be a drastic example but it gets the picture across. Real-time operating systems have been around for a long time and contain a foundation for processing interrupts in a timely manner with known results. Some of the most successful real-time operating systems such as MPX from Gould, RTOS from DEC, and others are used for flight simulators and operating mission critical parts of the NASA space shuttle. If tighter real-time specifications were needed, then the coding might depend on the assembly language level of software. One can see many levels and generations of software applied in the NASA shuttle program ranging from FORTRAN and ADA, to assembly language. X-Windows and Microsoft Windows may not match up to the specifications for fast interactive graphics. A software engineer should acquire experience on any and all types of operating systems and understand where they fit, or do not fit. Programming languages also fall into the same type of criteria for project requirements.

A BRIEF HISTORY OF GUI AND X-WINDOWS

A GUI is an interface to allow a user to communicate with an application on a computer. The interface is made up of dots arranged on a Cathode Ray Tube (CRT) in such a fashion that the dots represent a picture frame which is called a window. This is something that is taken for granted today, but in the 1960s through today, it has been an evolving technology. The CRT technology evolved from black-and-white to color. Another advance in the display technology has been the creation of thin-client nodes. A thin-client node is a graphic display with memory and a processor added. The node does not normally have a disk drive. The node can be located on a network far from the computer application that is driving the graphic display.

A user-interface in Information Technology (IT) terms is any method used to converse with a user from a software program. Software architects have always searched for the most effective method available. The user-interface found from the 1960s through the 1970s normally used a text device, called "dumb terminal." The "dumb terminal" was either a teletype type of device or a CRT that supported only text presentation. Graphic terminals for general use were not available yet; a programmer was stuck with available technology.

During the 1970s, UNIX was the foundation for X-Windows development. Companies like Xerox were also working on ideas similar to UNIX and X-Windows. Xerox produced the Alto computer in 1973 and was the first computer to use a GUI; the Alto was more of a research tool than a commercial product. The Alto can take claim to influencing other technology advances with respect to the GUI, mouse, and network development [16]. Apple Computer's Steve Jobs visited Xerox in 1979 which led to the GUI and mouse being integrated into first the Lisa computer followed by the Macintosh [16]. The GUI concept began to catch on as an effective method for a user interface. Another claim that Xerox bears is inventing Ethernet. Ethernet was developed in the period of 1973-1975 and was patented by Xerox in 1975 [5]. Robert Metcalfe was one of the inventors of Ethernet and had other ideas for the technology. Metcalfe left Xerox in 1979 to form 3Com and convinced DEC, Intel, and Xerox to promote Ethernet as a standard which was published in 1980 [5]. The software and hardware technology used to make up the Ethernet foundation was important for the success of X-Windows.

Next was the spread of the UNIX software to universities, Massachusetts Institute of Technology (MIT) being one of them. MIT produced the first version of X-Windows in 1984 through a collaboration of various people that had worked on similar types of projects in previous years. As an example, the Athena project was a joint venture between DEC, MIT, and IBM. The Athena project was to provide easy access to computing resources for all students [11]. One of the first computer

companies to officially use X-Windows in their workstations was Sun Microsystems. Sun was conceived in 1982 [13] and had been instrumental with DEC and IBM in further development and enhancement of the X-Window technology. DEC introduced the VAX computer in the 1978-1979 timeframe with the VMS operating system and introduced DEC-Windows for VMS workstations; DEC-Windows used a version of X-Windows foundation. Many graphical software products had inspirations from the Athena project which spanned from 1983 to 1991 [1]. By the early 1990s, X-Windows in itself was the product of the culmination of many technologies coming together over a span of twenty years.

The X-Window system is also known as X11 or just "X" in computing circles. The evolution of X11 depended on UNIX, bit-map displays, GUI development, Ethernet, and, of all things, the mouse. Scientists at Bell Labs experimented with different operating system (OS) concepts in UNIX so as to differentiate them from the IBM legacy operating system paradigm. Software languages offered by IBM at that time were COBOL, RPG, Assembly, and FORTRAN. Bell Labs in turn created a language called 'B', later changed to 'C' which is the language that a majority of UNIX is written in, with the remainder of the UNIX written in assembly language.

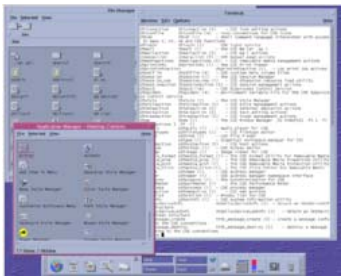
UNIX was developed to help scientists utilize the computer for scientific research. UNIX introduced new OS concepts such as stack architecture, events, processes, symbiotic drivers, and memory management. The UNIX OS concepts forged the work that would aid in the X-windows development. Much network theory and concepts, such as SENDMAIL, BIND, DNS, and DHCP also derive their background from the early UNIX development.

The Defense Advanced Research Projects Agency (DARPA), an agency of the United States Department of Defense, is responsible for new technology for use by the military. DARPA was responsible for the funding of the first beginnings of computer networking; the first result ARPANET, eventually grew into the Internet [4]. DEC approached Rice University in the 1970s to design and build the first Ethernet controller for use on the PDP11 series of mini-computers, then later on the VAX. It was the basis for the VAX network and DECNET protocol used in the 1980s. The introduction of sockets was developed using concepts derived from Ethernet theory. The Internet socket, commonly called a network socket or just "socket," could be considered the base foundation for future development of socket theory in UNIX [8]. The network socket provided a unique communication end-point associated with an Internet network protocol. In parallel, Berkeley sockets were first introduced in 1983, which was one year prior to the MIT release in 1984 of X-Windows to the public. The socket technology has played an important role in OS development such as inter-process communication (IPC) in UNIX [15]. Universities such as Rice and Berkeley took the new technologies such as sockets and applied them to innovative ideas of OS theory, such as Named-pipes, Datagram sockets, and Stream sockets. Today, socket programming is an important topic in Microsoft and UNIX programming. The Winsock reference can be found in various books written for Microsoft programming. Bob Quinn and Dave Shute wrote a book called *Windows Sockets Network Programming* which introduces Winsock versions 1.1 and 2.0 for Microsoft Windows. A web site, http://www.sockets.com, is promoted by Bob Quinn to help programmers learn socket programming in UNIX and offers code snippets.

X-Windows is written with the UNIX socket methodology under a client-server operation. The socket mechanism was first introduced in the 4.2 BSD UNIX system in 1983 in conjunction with the TCP/IP protocols that first appeared in the 4.1 BSD UNIX system in late 1981 [2]. Socket operation is transparent to an X11 programmer because the X-library routines handle all socket operation, hiding the X-Client interaction to the X-Server in an X11 application. The X-Server can exist on the same computer as the X-Client or the X-Server can exist across the network on a different computer. Because the actual location of the X-Server is transparent, it does not prevent the programmer from using socket connections to other applications. The X-Server handles all end-points. Douglas A. Young [18] described the use of an X-based mail program, which was the client, on a workstation in California against the X server which ran on a workstation in England. The response suffered due to the satellite transmission time, but X worked perfectly.

Microsoft produced the first release of UNIX on a PC, called XENIX, in 1980 well before the release of MSDOS. Microsoft did not have any focus or interest yet on Ethernet. In 1983, XENIX was ported to the Intel 8086 processor by the Santa Cruz Operation (SCO) and eventually became SCO UNIX in 1989 [14].

Well known companies worked diligently in the 1980s to make X11 a product that was platform independent. Several iterations of X11 software enhancements evolved with the client-server model pushing it forward in technology terms. X11 had been freely available to everyone; therefore, X11 became the first large-scale free software project [17]. Universities and companies together combined to make X11 a viable tool for the workplace. To keep X11 from fragmenting in the market place, vendors convinced MIT to maintain a neutral party ownership of X11 and lead the MIT X-Consortium, a non-profit vendor group of which several leading companies maintained membership [17]. A proposal was put together to create the MIT X Consortium as an open organization to be funded by the participants with a charter of supporting and controlling the development and evolution of the system. The MIT X Consortium was created in January 1988 [11]. In 1993, X-Consortium, Inc was formed as a successor to the MIT X Consortium. In the timeframe between 1993 and 1996, the X-Consortium released an X-Windows Motif toolkit and the Common Desktop Environment (CDE) for UNIX systems [17]. CDE is written using the Motif widget toolkit. Motif refers both to the GUI specification and the widget toolkit for applications written in X-Windows and is the basic building block of CDE [3]. Figure 1 displays a good presentation of CDE in action.



CDE, a graphical desktop for UNIX, was co-developed in the 1990s by HP, DEC, IBM, and SUN.

Figure 1 – Common Desk Environment (CDE)

PROGRAMMING X-WINDOWS AND COMPARISON WITH MICROSOFT WIDOWS

Here we give an introduction to programming considerations when developing a program using X-Windows. Application development in X-Windows requires an understanding of how X functions during execution. The approach presented will guide a programmer through the design and development stages of a project. The assumption is that if you understand how X works, one can perform a better design of an application that uses X as an interface. Throughout, we will distinguish how the methods would differ in the more well-known Microsoft Windows.

During execution, the X-Client is required to perform an activity with the X-Server in three separate steps. The first step is to define the main window to the X-Server. Parameters are presented to describe the characteristics of the main window. The second step defines the components and resources to assemble into the defined window. This step differs from the Microsoft method in that the components in X are built and placed on the window form on-the-fly during execution, whereas the Microsoft components are added to the form during code development and fixed at compile time. The X-Window method can be modified using outside parameters to influence the changes to the program without re-compiling. This offers a user the opportunity to modify the look-and-feel of an X-Window without access to the source code. The third step of X execution gives commands to the X-Server to render the X-Client window. The X-Server then enters into a listen-mode waiting for resource events. All component resources have been presented and armed in wait for a corresponding event.

The first step in coding an X window is the definition phase. The definition code will create all aspects for the main display window and any sub-windows to the X-server. The definition code can be small and simple or large and complex based on the X-Window design requirements. The simple case is just to specify to the X-server that a window of a certain size is required. A token or handle will be provided by an X routine that creates the display for the window. Further code written referring to this window must use this token. The token is a pointer to the table logic held internally for the corresponding window. Characteristics and resource information is held in this table. X routines manage this table during execution. In the Microsoft Windows environment, Windows also return a handle which a programmer uses to conduct communications with the window. The definition code can provide the type and size of font to use, any color for background and borders, and other types of X-Server handling such as resizing or placement of the window on the screen. If no characteristics are given, X-Windows will use a default set of characteristics, sizes, and colors. Special files exist to support X applications during execution. These files can be modified by a user to alter the characteristics of the window. A command is required by the user to force the X application to re-read the external files and apply changes. This feature does not require modification of the X source

code or a recompile. Microsoft does not support this capability and requires recompiling the program after source code modification to affect window characteristic changes.

The second step in coding an X window requires code to define components and resources to insert into the main window. This code places buttons, text boxes, labels, and any other type of components into the main window. These components are considered resources of the main window. Sub-windows or pop-up windows belonging to the main window are called children of the main window and are created following the same steps as the main window, the difference being that the token of the main window is passed as a parent token to the children of the main window. The main window is an anchor for children pop-up windows. A child pop-up window can also have children pop-up windows. It is important to remember this stair-step architecture for parent-child association of windows because when a window is destroyed, all of the children of that window also go away.

The last step in coding an X window requires code to command the X-server to invoke, render, and manage the window and all resources. The purpose of the X-server is to accept requests for graphical output to the window and send user input from the keyboard, mouse, or touch-screen to the X-Client.

X11 code is written in C. The X and Motif Libraries are callable by C routines. Routines written in C to create X-Window screens are able to connect to routines written in languages such as Pascal and FORTRAN. The underlying application can be written in most programming languages and linked to the X code. For most cases, the entire application is written in C, but may not always be the case. We have observed several projects which consisted of a mixture of modules written in several languages calling each other; each language had a value that the others lacked..

It is important to point out that one of the valued features sought in operating systems and languages was the introduction of a subject called "stack oriented" operation. The theory of the operation of stacks allows the saving of information upon entry to a routine. A routine could be entered several times before any exits could occur. The order of exit would match the order of entry and matches the theme of "last-in, first-out"; each exit would cause a pop from the stack and would match the push to the stack caused by the entrance of the routine. The pushing and popping of information from the stack is an important software technology that opened doors to future application solutions and did not require any special code. The C language was designed with this feature allowing C routines to be re-entrant by nature. A majority of UNIX is written in C giving UNIX the benefit of stack structures throughout and gives UNIX capability to support "almost" real-time application support capability. The stack oriented feature is now found in most operating systems today and have their origin from UNIX. VMS took this feature further and made all languages used under VMS re-entrant.

The X-Server plays an important part and may function as one or as all three of the following operations: (1) the first operation could be an application displaying to a window of another display system, (2) the second operation could be to act like a system program controlling the video output of a PC, (3) the third operation could be to act like a dedicated piece of hardware. The inter-process communication is provided by X libraries routines are transparent and allow the X-Client to perform independently of the X-Server. More information about each X library capabilities can be found in the various programming books for X-Windows and Open System Foundation (OSF) design guides. The Motif programming guide is also available and provides fancier look-and-feel visuals in X. The Motif library is routines that will produce components in three dimensional form for effect. Because of this, the Motif library can be a little more difficult to use.

When designing X-Window applications, the planning and design of the hierarchical structure of the application is important and can make the code easy to manage and maintain. The following design criteria were applied on a software project at a seismic service company:

The goal was to keep code separate from each other based on function and the hierarchy of the design specification. As an example, the database section of the project was kept separate and independent from the other project sections. This hopefully would prevent human errors that may occur from mixing source code. If there were routines shared by other sections of the project, they were moved to an area for maintaining common library support. During the move, an analysis of the code would determine the naming convention for the routine and modifications made to reference the new name where applicable. Each library would be maintained in the corresponding directory based on function. A common directory would hold all libraries. Application "make" files would point to

this library directory for linking. Code was maintained using the SCCS utility in UNIX.

The naming of each main program file would be named so as to associate it with the purpose of the program. Appending the string prefix "app_" to the name separated the file from subroutine and function files. As an example, the database program had a file name of app_dbManager.c and contained the C language "main()" statement. X-Window programmers normally follow the standards set by the X programming guides and the examples of code snippets provided. The examples may show either the Hungarian notation and/or the Camel notation method of naming convention or variations thereof for routines used.

Each "app_" program file contained the "main()" statement plus supporting application routines used by the program.

The naming convention in (2) also applied to subroutines and functions used by the program, but did not have the "app_" prefix. A special string prefix associated the routine with the program. As an example, the database routines used the prefix "db_".

Code required to define and support a popup window routine would exist in the file for the corresponding pop-up window. This included any callback routines for the associated pop-up window. Examples of the types of code in the file would include component definition calls, callbacks, and routines written for that pop-up.

A callback routine name would associate it with the corresponding window component such as a button. By selecting a window button would cause the callback event-handler entry for that button. A name example would be btn_OpenDatabase_CB(). Going from left to right, the "btn_" is a prefix for all button routine names followed by the purpose of the button "OpenDatabase" then the action of the button "_CB" which indicates that the routine is a callback event handler. The database management program had the option of opening any one of several databases. This button caused a popup window to present a list of available databases allowing the user to select and open one.

These are just suggestions for naming conventions and can vary as needed. If such a set of conventions is applied, it allows for easy interpretation of the source code, thus facilitating future maintenance. If each person uses their own method of programming, it may or may not be understood by others. Programming style must be a committee effort to specify and should be adhered to by all, or else chaotic source code will result. The ideal methodology is one everyone can understand, which includes inexperienced and experienced programmers. Without good naming conventions, X-Window programming can become chaos in a hurry and becomes difficult to follow and maintain. This holds true for structures, variables, routine names, and window references.

The event handler, also called a callback routine, is important in GUI programming. An event handler is created differently in X-Windows than in Microsoft Visual Basic (VB) programming. For instance, an event handler routine is created automatically by Visual Basic when a component is added to the VB window form. The name of the handler would have the word "click" as part of the name if associated with a button. The VB event handler routine is an empty shell and requires a programmer to insert the proper code into each empty shell to complete the function of the handler. The VB application can be compiled with no indication that the body of code for an event handler is missing. By contrast, an event handler in X-Windows is called a callback routine and must be written by the programmer and presented to the X-Server when a component is defined.

Another difference between Microsoft Windows and X-Windows is the method of component placement within a window. The method in X development is performed by specifying either an exact X-Y coordinate position or a relative coordinate position with a subroutine call. The coordinate is passed as an argument in a routine call to define the component. The location placement is a burden to the programmer and can be a case of mental gymnastics and experimentation when developing a window in the X environment. Once the trick of the X-Y positioning system is learned, it is a piece of cake. It can also be amusing to view what was created and a puzzle to discover what code or parameter caused it. The method for component placement in Microsoft Windows is more of a drag and stretch effort on the VB form during coding.

X-WINDOWS ENVIRONMENT AND MICROSOFT WINDOWS

The X-Window development environment does not provide a visual development package like the Microsoft Visual Studio. All X-Windows development must be created using UNIX editors like *vi* or *emacs*. For an experienced programmer, the X-

Windows programming environment is fun to work with, yet challenging. A must for a programmer is the understanding of the C language, UNIX, the X-Window programming guides, plus any acquired X programming books.

X-Windows is a well defined mature system for developing user interfaces for applications under UNIX and Linux. Linux also supports the X-Window environment and can accept applications from UNIX systems and vice-versa with minimal modification to the source code. Another benefit with UNIX and X-Windows is the availability of the thin-client computer node. The X thin-client node is a computer that may be diskless and boots a small simple UNIX OS from a host on the network. A thin-client node provides an inexpensive method of allowing a user to log onto a UNIX server on the network and have graphic capability. Both UNIX and Linux systems support a multi-user environment. This means that several thin-client nodes can be connected to a UNIX system performing work, all nodes using X-Windows across the network.

An interesting feature of the X-Window system is that X-Window graphics output can be redirected to a different UNIX or Linux system on the network. A Microsoft Windows node is also a candidate to receive X11 graphic drawing commands if the Microsoft node is running *Interix* or *Reflection X*. A user can cause this redirection from an X-Server using an environment variable that specifies the IP address of the target node. Of course, the target computer must have permissions set to allow the receipt of redirected graphics.

INTEROPERABILITY

A program called *Relfection X* was written for Microsoft Windows and distributed by Digital Equipment Corporation in the 1990s. *Relfection X* is a software product to promote UNIX interoperability [10]. It is an X11R6.9 PC X server which connects Windows users to graphical and character-based applications on UNIX, Linux, and OpenVMS hosts. Hummingbird Connectivity, a division of Open Text Corporation, sells *Exceed*, a competitive software package for the PC X Server market for users of Microsoft Windows [6]. Another product also existed in the 1990s called *Nutcracker* from a company called DataFocus, Inc. [9]. *Nutcracker* and *Interix* are just a few products that supported porting UNIX programs to Microsoft Windows. Users also use these products to support X-Window applications and conduct NFS connections on the network. Most of these products are still available for Microsoft users that require connection to UNIX systems on the network.

Interix came from a company called Softway Systems, Inc. and was marketed in the 1990s under the name of *OpenNT*. *OpenNT* had a Software Development Kit (SDK) which allowed porting of UNIX programs to Microsoft Windows. *OpenNT* also supported X-Windows rendering just like *Reflection X*. OpenNT caught the attention of Microsoft and was acquired by Microsoft in 1999. Microsoft renamed *OpenNT* to *Interix* and started integrating the SDK into the Microsoft product lines [12]. *Interix* can be found in some releases of Microsoft Windows as a component of service. Release 3.0 was delivered as a component of *Services for UNIX* (SFU) in 2002. Version 3.5 was delivered later and distributed free. *Interix* is integrated as a component of Windows Vista and will be integrated in Windows Server 2008. *Interix* provides over 350 UNIX utilities such as vi, ksh, csh, awk, grep, kill, and many others. *Interix* also supports the X-Windows client applications and libraries [7].

X11 PROJECT TOOLS AND REFERENCES

Application design and X-Window development for any project can be fun and challenging, especially when helpful snippets of X code samples could be found. Examples of X-Window code snippets are found in most X-Window and Motif books, some are accompanied with CDs. Another good place to find X-Window source code to experiment with it on websites on the Internet. University websites usually provide some of the best discussions, papers, and code examples on X-Windows. A recent search for X-Window code generators found a broad range of hits covering all kinds of subjects on X-Windows, some related to code generators. Materials found on the Internet are helpful in learning how to create X-Window screens, components, and how to manage them. The following is just a few interesting websites located from a recent search:

http://www.labf.com/
http://www.interopcommunity.com/default.aspx
http://www.scit.wlv.ac.uk/~jphb/comms/sockets.html

The following are five volumes available from any book dealer. This list can be found on the O'Reilly web site http://www.oreilly.com/catalog/motifref2/.

Figure 2 – Manuals on X-Windows

The manuals are over a foot high when stacked in a pile and contains more information about X-Windows than could be discussed in this paper. These books are normally in a set of five. Other books on the subject are also available, but these are the most common. Some colleges offer courses in X programming which require pre-requisite courses in UNIX.

SUMMARY

The X-Window graphical user interface is an interesting product with a rich technology history behind it. It is important to understand this historical path to help understand where and how to utilize the best and appropriate solution using X-Windows in application work. The history behind X-Windows helps a person with a software career understand where most of the technology came from that they may touch in projects. Technology history also exposes tools and resources that may also be useful. No stone should be left unturned lest there be gold under it. The same goes for ignoring information about important software disciplines including Microsoft Windows, UNIX and X-Windows. Fortunate is the programmer who has touched all of the above in their career in software projects. The information put forth here may help someone with a brief exposure to the world of X-Windows to understand the rich technology foundation that is available for future projects.

This discussion is more about the technology that people invented during the progress of Unix and X. These technologies are now found in a lot of products that are not related to UNIX or X. It spurned the growth of computers into areas that would never have happened if not for UNIX and the people that made it happen. VMS and Microsoft operating systems would not exist if it were not for UNIX.

REFERENCES

1. Athena. Available from: http://en.wikipedia.org/wiki/Project_Athena. [Accessed June 3, 2007]

2. Burden, Peter. 1999. Sockets programming. Available from: http://www.scit.wlv.ac.uk/~jphb/comms/sockets.html. [Accessed June 3, 2007]

3. Common Desktop Environment. Available from: http://en.wikipedia.org/wiki/Common_Desktop_Environment. [Accessed June 16, 2007]

4. DARPA. Available from: http://en.wikipedia.org/wiki/DARPA. [Accessed June 3, 2007]

5. Ethernet. Available from: http://en.wikipedia.org/wiki/Ethernet. [Accessed June 3, 2007]

6. Hummingbird Connectivity. Available from: http://connectivity.hummingbird.com/home/connectivity.html?cks=y. [Accessed June 3, 2007]

7. Interix. Available from: http://en.wikipedia.org/wiki/Interix. [Accessed July 15, 2007]

8. Internet Socket. Available from: http://en.wikipedia.org/wiki/Internet_socket. [Accessed June 3, 2007]

9. Nutcracker. Published in Software Magazine, April, 1995. Available from: http://findarticles.com/p/articles/mi_m0SMG/is_n4_v15/ai_16864570. [Accessed July 15, 2007]

10. Reflection X. Available from: http://www.attachmate.com/en-US/Products/Products.htm. [Accessed July 15, 2007]

11. Scheifler, R. W., Gettys, J., & Newman, R. 1988. Introduction: history. *X window system: C library and protocol reference* (pp. xxiv-xxix). Maynard, MA: Digital Press.

12. Softway Systems. Available from: http://www.microsoft.com/presspass/press/1999/Sept99/softwayPR.mspx. [Accessed July 15, 2007]

13. SUN Microsystems. Available from: http://www.sun.com/aboutsun/company/index.jsp. [Accessed June 3, 2007]

14. UNIX. Available from: http://en.wikipedia.org/wiki/UNIX. [Accessed June 3, 2007]

15. UNIX domain socket. Available from: http://en.wikipedia.org/wiki/Unix_domain_socket. [Accessed June 3, 2007]

16. Xerox Alto. Available from: http://en.wikipedia.org/wiki/Xerox_Alto. [Accessed June 3, 2007]

17. X Window System. Available from: http://en.wikipedia.org/wiki/X-windows. [Accessed June 3, 2007]

18. Young, D. A. 1990. An introduction to the X window system: The client-server model. *X window system: programming and applications with Xt* (pp. 2-3). Englewood Cliffs, NJ: Prentice Hall.