BEST TARGET PLATFORM FOR APPLICATION MIGRATION

[1]MAHDI MALEKI , [2]SHUKOR ABD RAZAK

[1]Faculty of Computer Science and Information System, UTM, Skudai, Johor, Malaysia-81310

[2] Dr., Department of Computer System and Communications, UTM, Skudai, Johor, Malaysia -81310

E-mail: mmahdi3@live.utm.my, shukorar@fsksm.utm.my

ABSTRACT

This paper aims on finding common key factors between different platforms that must be considered in cross-platform application migration and the assigning of weight to each of individual factors. These are achieved through two surveys. This paper will propose a standard method /.for compatibility ratio measurement regarding the origin/source and destination platforms to help system administrators and IT managers to choose the best platform in migration projects.

Keyword: *C++, Application Migration, Cross-Platform, Migration Issue, Legacy Application*

1. INTRODUCTION

According to the latest surveys about programming language popularity, most companies deciding to develop an enterprise application preferred to use one of the cross-platform programming languages for development [28]. Most of the reasons given for using cross-platform programming language are related to their prominent features such as flexibility and portability. Furthermore, the future dictates a need to cater to a growing number of application users, the changing of security policies and so forth that maybe compel IT managers to choose another platform to achieve more performance, extra capabilities, security enhancement, decreasing TCO (Total Cost of Ownership) and increasing RAS (Reliability, Availability, Serviceability)[14].

As Richter et al. [22] states, choosing the most suitable and compatible target platform are one of the big challenges of every project migration process. Accordingly, this paper will propose a standard method to evaluating competitive ratio between the source and destination platform to decrease the risk of application migration between two heterogeneous environments.

According to Wilson et al [30] around more than 50% of migration project after two to four years are abandoned due to failures that cost lots of money and wasting resources and time. This kind of migrations usually is happened between UNIX and Linux based platforms [13].

TABLE 1. HISTORIC PROGRAMMING LANGUAGE POPULARITY RANKING [17]

Language	10-Sept	06-Sept	00-Sept	85-Sept
C	1	2	1	1
Java	2	1	5	n/a
C++	3	3	2	10
PHP	4	4	31	n/a
(Visual) Basic	5	5	3	4

2. APPLICATION MIGRATION/PORTING

Migration can be defined as the process of porting from/to another heterogeneous /homogeneous operating environment. Usually, it is considered as movement to a better environment. For an instance migrating from Windows NT Server to the newer version, Windows 2000 Server may be considered as a migration project because there are some new features that one could benefit while maintaining old configurations do not need to be changed; it also involves steps to make sure that

current applications will be operational in the new environment [3].

Every single cross-platform application migration procedure can be divided into following steps [10]: Planning and detailed assessment, Tools for development and customization, Test migration, Application migration, Acceptance, Installation, warranty, and product support.

According to Bierhoff, et al [1], every single IS application can be defined as a series of components plus communication between components. Components can be divided into four subcategories [1]:

- Functionality supply
- Infrastructure expectations
- Control model
- Data manipulation

Additionally, communication between components can be classified into two sub-categories [1]: Asynchronous communication and Message data model.

There are two well-known scenarios for doing migration (porting) [5], which are:"Port and Modernize", and "Modernize and Port". The "Modernize and Port" scenario initially identifies the value of existing application, and then tries to migrate the application by porting some parts of the code while extracts the business rules. Usually, this includes the core applications which are very important for a large number of corporate users, as it can drive critical business processes in a very adequate way. However, there are disadvantages to this method, which includes the difficulty in comparing between new and old running performances, a complex process of testing and acceptance [5].

The "Port and Modernize" scenario split the project into different portions of codes migration and then recognize components by doing separation between port and modernization. As Intelligent Business Solutions [5] suggested,"Modernize and Port" scenario is much feasible than the other scenario.

3. CURRENT ISSUES

During each phase of application migration, there may be issues at different layer of application. To go deeper into this and find coexisting incompatibilities, it is better to categorize these issues by layers of application. The application can be divided into six layers:

- Presentation Layer
- Application Service Layer
- Lending Message Bus Layer (optional)
- Business Layer
- Data Service Layer
- Platform Service Layer

3.1. Presentation Layer (PL)

This layer also is called user interface (UI). Three most common types of user interfaces are: Graphical User Interface (GUI), Web-based User Interface (WUI), and Command Line Interface (CLI). One of the most outstanding issues is font incompatibility which means font(s) that used in ported application is not supported by destination environment.

TABLE 2. COMPARATIVE DEPENDABILITY OF FONTS FOR WEB-USE: ORDERED FROM MOST TO LEAST FREQUENTLY FOUND IN WINDOWS [17]

Font	Windows	Mac	Linux
Arial Black	97.73%	95.67 %	54.44 %
Verdana	97.41%	94.02 %	55.00 %
Arial	96.97%	96.41 %	62.78 %
Courier New	96.79%	92.08 %	61.94 %
Comic Sans MS	96.72%	91.63 %	51.94 %
Lucida Console	96.65%	n/a	n/a
Tahoma	96.61%	72.50 %	n/a
Impact	96.33%	88.04 %	53.89 %

Statistics about font compatibility among Windows, Mac and Linux is as shown in Table 2. The other possible issue is when application use UNICODE characters when destination operating system does not support UNICODE as in some UNIX-based operation systems. Another possible issue is browser incompatibility (See Table 3).

3.2. Application Service Layer (ASL)

Reusable components are placed in this layer and the functionalities represented at the presentation layer are provided by this layer. This layer plays the role as mediator between business layer and presentation layer through the "Lending Message Bus Layer". In this layer, type casting and type mismatching issues are very common.

Usually input data which is received by presentation layer should be casted. Casting issues generally happen when application is ported form 32-bit environment to 64-bit operating system due to having different data models (See Table 4).

TABLE 3. BROWSER COMPATIBILITY CHART [4]

Browser	Windows	Linux	Solaris
Flock (1.x and 2.x)	Yes	Yes	No
Internet Explorer	Yes	Yes	Yes
Konqueror	Yes	Yes	Yes
Mozilla	Yes	Yes	Yes
Netscape Navigator 9	Yes	No	No
Netscape Browser	Yes	No	No
Opera	Yes	Yes	Yes

3.3. Lending Message Bus Layer (LMBL)

Messages can be traversed through the Lending Message Bus Layer in two ways: Asynchronous and synchronous. Most outstanding consideration in this layer is the supported communication protocols and how network resources are accessed by the application. Moreover, different message models can cause incompatibilities in the formats of messages exchanged by components. This can lead to massive performance overhead due to costly message conversions.

TABLE 4. SHOWS SIZE OF EACH TYPE IN DIFFERENT MODELS [15]

Type-Model	ILP32	LP64	LLP64	ILP64
char	8	8	8	8
short	16	16	16	16
int	32	32	32	64
long	32	64	32	64
long long	64	64	64	64
size-t	32	64	64	64
pointer	32	64	64	64

3.4. Business Layer (BL)

Most of migration issues happened in this layer because the majority of low-level programming and business logics implementation is coded in this layer. As another point of view the core and most critical part of application is in business layer. The first issue is using function with variable number of arguments.

Using magic numbers in address calculation, bit operating, endianism and object size during a migration are another example of issues in this layer.

File name restriction is the other common issue especially when migration is happening between two heterogeneous operating systems like Windows and UNIX-based Operating system. Another consideration is about ACL or file security and access permission. (See Table 5)

3.5. Data Service Layer (DSL)

Data service layer is responsible for providing data for business layer which is also known as Data Access Layer (DSL). The main problem in this layer is with drivers and their used protocols for connecting to different data sources. Data sources can be vary from simple text file to complex relational databases.

TABLE 5. FILE SYSTEM COMPARISON CHART [11]

File System	Case-sensivity	Reserved characters	Max. length	Max. File size
MS-DOS FAT	case-insensitive, case-destruction	any	12	4GB
NTFS	optional, case-preservation	any (include UNICODE)	255	2^64 bytes
UNIX	case-sensitive, case-preservation	any	255	2^73 bytes

There is very simple differences between texts file structure in Windows-based operating system and UNIX-base one which is line separator. UNIX files use a linefeed (LF) character for line separation, while Windows files use carriage return and linefeed (CRLF).

Application should use driver that provide connectivity to data source. Open Data Base Connectivity (ODBC) is most common driver which uses as this purpose. UNIX-based operating system does not support ODBC calls, so third-party vendors' application such as DataDirect should be used as an alternative.

3.6. Platform Service Layer (PSL)

Platform defines as combination of hardware and operating system together while hardware usually

knows as system architecture. Three most famous architectures are: X86, SPARC and IA (Intel Itanium). All these three architectures have own specific consideration.

Porting from X86 to IA has minor issue that performance of application that is compiled at X86-32 and ported to IA-32 is decreased dramatically.

Operating system is another part of platform has many flavours. Four major streams of operating systems are UNIX, Windows, OS2 and AIX. They have differences fundamentally: supported file systems, kernel types, memory management, native APIs and resource access control are just part of these differences [11]. Table 6 and Table 7 show differences between six most common operating systems.

Supported character sets by operating system are another issue. This issue impact on both presentation layer and also business layer. Legacy operation systems use ASCII standard as their default internal operating system encoding because of variety of using language-specific extensions of ASCII which made exchange files difficult.

TABLE 6. OPERATING SYSTEMS DIFFERENCES (PART I) [12]

OS Name	Supported Architecture	Supported File System
AIX	PowerPC	JFS,JFS2,ISO 9660,UDF,SMBFS,GPFS
Linux	x86,x86-64, ,PPC,SPARC ,Alpha	ext2,ext3,ext4,ReiserFS, FAT,ISO 9660,UDF,NFS
HP-UX	PA-RISC ,IA-64	VxFS,ISO 9660,HFS,UDF,NFS,SMB FS
Solaris	x86,x86-64, SPARC	UFS,ZFS,ext2,FAT, NFS,QFS,NTFS,FAT,exF AT
Windows Server 2008	x86,x86-64 ,IA-64	NTFS,FAT,exFAT, ISO 9660,UDF, ext2,ext3,reiserfs9,HFS+,F ATX, and HFS

While UTF-8 widely uses text files (source code, HTML files, email messages, etc.), file names, standard input and standard output, pipes, environment variables, cut and paste selection buffers, telnet, modem, and serial port connections to terminal emulators, some operating systems are not support UTF-8 character set by default and you have to install it first.

TABLE 7. OPERATING SYSTEMS DIFFERENCES (PART II) [12]

OS Name	Native APIs	Non-native APIs	Resource access control
AIX	SysV POSIX	n/a	Unix,ACLs
Linux	POSIX	Mono,Java, Win16, Win32	Unix,ACLs, MAC
HP-UX	SysV POSIX	n/a	Unix,ACLs
Solaris	SysV POSIX GTK,Java	Win16, Mono,Linux ,Win32	Unix,RBAC ,ACLs, Trusted Extensions
Windows Server 2008	Win32,NT API	.NET,Win16, DOS API, POSIX	ACLs, Privileges RBAC, Least Privilege

There are some special languages which have to be considered as high risk challenges if they used in application. Languages such as Arabic, Chinese, Korean, Persian, Thai, and Malay are just few examples of high-risk languages which must be highly considered before doing migration [7].

Generally porting one application between two completely heterogeneous platforms makes lot of challenges. One of the main challenges is administration of new platform which needs to learn many things for old administrator or hire costly senior system administrator to manage new environment. Sometimes new packages should be bought to fulfil administrative tasks such backup utilities, antivirus, firewall and in some cases licenses of current application must be upgraded to support new environment.

These challenges include [12]: Disk Administration(Partition Disk, Creating Stripe/Volume sets, Remote Disks), File Services(File Security, File Names, Sharing Files), Printing Services(Printer Sharing, Remote Printer, Adding/Changing Printers, Printer Drivers), Communication Services(TCP/IP Setup/Changes, FTP, TFTP, RCP, and Telnet, E-mail, Talk/Chat, User Messages), Backup/Recovery, System Log Files, Process and Task Management (Task Priority, Automatic Job Scheduling, System Load Balancing, Terminate Process/Task, Daemons/Services, User Management).

4. RELATED WORK

This section discuss about what are the existing solutions for issues listed in previous section. Accordingly possible solution will represented based on layered architecture which is introduced in this paper. While Wojtczyk, et al [20], introduce a development framework for cross-platform C/C++ applications that applicable for rebuilding importable modules. Understanding the different type of changes is essential which helps knowing how these issues created and then go through the some solution for mentioned issues in different layers.

Changes can be categorized in three sections [14]: Functional changes, Non-functional changes, Platform changes. Indeed in many cases platform changes made major issues but in some cases other types of changes made minor issues which can be called incompatibilities at functional changes and coincident at nonfunctional changes.

Platform changes can be divided into two categories regarding to their impacts: platform minor changes, platform major changes.

4.1. Presentation Layer (PL)

User Interface (UI) is one the most important part of each application that represents output of the whole system. Hence have a consolidated UI helps to have well represented application. Application Programming Interfaces (APIs) play an important role in this layer. One way to reduce dependencies is using standard APIs besides using some generalization techniques such as using XML to keep configurations and common standard fonts at presentation layer [2].

There some methods and frameworks to develop independent cross-platform GUIs such as Trolltechs Qt and Gnome Gtk+ toolkits. Microsoft develop [11] Interix to support X Windows system on its operating system. It supports "*xterm*", "*xlsfonts*", and "*xrdb*". It is compiled using X11R5 and X11R6 libraries to run with the corresponding servers. If the migration is happening in the vice versa direction which is discussed above, there are some open source toolkits such as WINE and Mono. Moreover there are some enterprise and commercial toolkit such Wind/U and MainWin Studio.

Mono is a cross platform, open source .NET development framework. Since Common Language Infrastructure (CLI) is able to host C++ compiled code on all supported platforms as long as the compiled code only contains Common Intermediate Language (CIL) instructions and not a mix of CIL and native code. Microsoft Managed C++ and C++/CLI compilers produce mixed-mode assemblies by default. Mixed-mode assemblies are experimentally supported only on Windows because native code is platform specific. It provides core APIs for .NET technology.

WINE is another toolkit to run windows application on Linux, BSD, Solaris and Mac OS X. It implements the Windows API entirely in user space, rather than kernel module. It has special C++ runtime library to support C++ codes.

To make Command Line Interface (CLI) portable, using a standard character encoding is very important which should be supported by destination platform. Considering about simple differences such as option which is start with "-" in WINDOWS-based operating system and "/" in UNIX-based operating system and also uses of some special characters such as reserved character and some reserved filenames that may not legitimate at ported environment.

Typically Web-based User Interface (WUI) is cross-platform module because it just needs internet browser which almost all operating system have at least one embedded internet browser. The main important consideration in such these modules is web page layout; web configuration and internationalization (language) which can simplify tackle them by using standards such as XML and HTML.

Another consideration is about fonts which is used in web pages, they should be available on destination platform specially encodings which make problem in generating response pages if they are not available on destination platform. For an instance even you use UTF-8 encoding in ported application, you have manually install specific locale in Solaris to support it. To give more flexibility to WUI, it is highly recommended to use XML which is originally design o provide plainness, generalization, and usability for web applications over the Internet. Extensible Hypertext Markup Language (XHTML) is new established technologies that makes HTML more extensible and helps to have more interoperability with vary data formats. According to Taleb, et al [16], there is some Patterns-Oriented Design Applied to Cross-Platform Web-based Interactive Systems which can helps developer to have platform independent application.

4.2. Application Service Layer (ASL)

There are some tools available to check the source code to find such these type casting and type

mismatch problems. "*lint"* is one of these tools that is available for finding suspicious and non-portable constructs in C source codes.

Another tool is "*cl"* command which is integrated with Microsoft Visual Studio to detect common coding errors in C/C++ source codes. It can find type casting issues between 32-bit and 64-bit data models that mentioned before. In an application which uses structure or union to pack data and send it to business layer via lending bus message, padding must be considered because the total size of structure or union has variation in different data models(32-bit and 64-bit) [14].

4.3. Lending Message Bus (LMB) Layer(optional)

Lending Message Bus Layer is responsible for acting role as mediator between Application Service Layer (ASL) and Business Service Layer (BSL) which can be in two ways: Asynchronous and synchronous. While application is ported from single-thread processing platform to multithread environment asynchronous message encounter with some conflicts, especially when multi-threading with strict prioritize scheduling paradigm is applied [9].

Another consideration in this layer is supported communication protocols and how sockets and network resources is accessible by application. UNIX-based operating system networking is based on three layers which each layer has loadable module in kernel, modules including: socket interface, protocol drivers and network-device drivers. For an instance, there are some differences at using of sockets between windows-based and UNIX-based operating system that should be considered. One of them is header file which should be included: *"sys/socket.h"* in UNIX or *"winsock2.h"* for Windows.

Window-based operating systems have two internal interfaces: Network Device Interface Specification (NDIS) that control network adapter cards, and Transport Driver Interface (TDI) which works as mediator between the transport and session layer of the OSI model.

Another issue is again refers to different endianism between X86 architecture and Solaris and PowerPC architectures. While Solaris and PowerPC architectures use big-endian. X86 architecture use little-endian which should considered when bytes is received via network. Remote Procedure Call (RPC) is well -supported by almost all operating systems.

4.4. Business Layer (BL)

This layer contains the most important part of application which is Business Logic. It also known as the core of application, moreover needs to have many considerations for applying any changes in this critical part of application.

Functional algorithms that are implemented in following layer mostly focus on handling information exchange between Data Service Layer (DSL) and Application Layer Service (ASL).

Most computational algorithm and also data manipulation logics placed here, hence discussing about possible issues at logical and low-level coding is essential. The same considerations as they are mentioned at Application Service Layer (ASL) must be considered also in this layer. For instance, there are some functions such as *"printf", "sprint", "scanf", and "sscanf"* which can accept variable number of arguments that should be threaten very carefully to avoid buffer overflow or showing incorrect result. It is recommended to use *"cout"* instead of *"prinf"* and *"sprint"*, and replace "sprint" with *"boost::format"* or *"std::stringstream"* [6].

4.5. Data Service Layer (DSL)

Data service layer provides data for business layer, so the main consideration here is about how it can persistently work on different environment. Database connectivity driver must be supported on both platforms to have steady data access service. Open Database Connectivity (OBDC) is one of these drivers which are mostly recommended by experts.

Because many database vendors try to make platform to support many platforms as it is possible we can totally say that there are serious problem with systems that use well-known RDMBS system in this layer. There just one major issue with system which put data into text files and migration is happening between UNIX-based environment and Windows-based operating system. This issue can be solved some available tools for converting text files.

Supported Locale or Language by operating system is another consideration, sometimes it needs to purchase and install new packages or changing configuration to support desire encodings.

4.6. Platform Service Layer (PSL)

Architectural mismatch leading to have different implementation of platform services and consequently distinctive may of resource and

service management. Memory and process management is one the well-known differences.

As it shown in Figure 1 and Figure 2, there are some architectural differences between Windows-based operating system and UNIX-based.

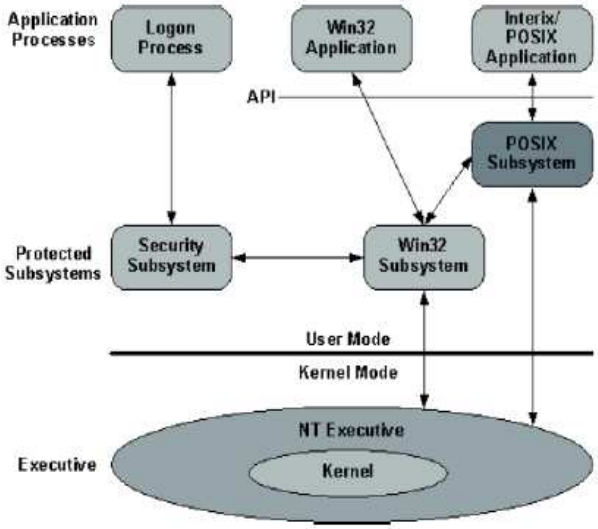

Figure 1. Windows Server 2003 Architecture [8]

It is strongly recommended if application is highly afflicted with platform and has many dependencies over its services to use Virtualization instead of migration that take many efforts and made a lot of challenges for both developers and system administrators to have successful porting project [18].

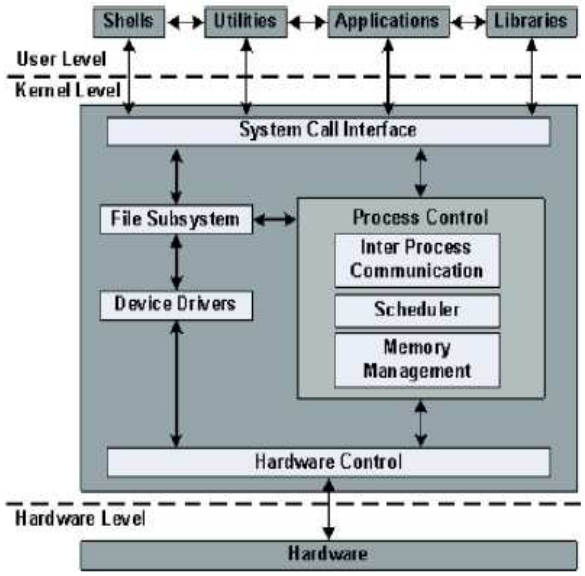

Figure 2. UNIX Architecture [8]

5. DISCUSSION

Nowadays, regarding to changes in IT environment including: size of application, capabilities, applicability, functionality and limitation of legacy applications. IT mangers might be deciding to migrate their enterprise application from inherited environment to new and fashionable platform to be beneficial of new technologies [14].

This migration usually is happened between UNIX and Linux based platforms. According to statistics around more 50% of migration project after two to four years are abandoned as failures which are cost lots of money and wasting resources and time [19].

This article aims to present some possible issues that might be affecting on ported application. Due to variety of issues, to find and tackle them in a feasible way, breaking application in different layer of concerns is proprietary. Hence, based on different services and functionalities of application modules, application is break down into five layers: Presentation Layer, Application Service Layer, Lending Message Bus Layer, Business Layer, and Data Service Layer. Platform Services which are represent by Operating system, its integrated modules and attached devices is another important consideration that sometimes makes big challenges for application migration project. This article, for the first time, defines Platform Service Layer (PSL) as new concept as part of IS system architecture that has not been consider in any pervious works. PSL is tightly affiliated by Operating system, correspondingly changes in operating environment sometimes has vast impact on services that is provided by this layer, even though if it is happening between two homogenous, still there are some functionality mismatches that should be concerned.

This paper aims to cover some common and well-known issues in different functionality areas by giving existence solutions while all previous works try to cover numbers of these incompatibilities without concerning about functionality, importance and severity of each individual issue on each layer of application.

Therefore, depend on type of application and interested area of functionalities, one of the two scenarios of migration can be chosen as feasible solution which can decrease defeat rate of application migration and also cost of implementation by having knowledge of possible challenges and issues before ahead. Lack of well-define framework to do migration for C++ applications is sensible.

6. CONCLUSION

During past two decades many legacy applications and mission critical appliances are developed based on C++ programming language. These applications run for many years and consolidated by removing bugs and exception which might take lots of efforts and resources to tackle them.

Due to vast changes and enhancement in IT environment and technological development porting application to new environment to be beneficial of these new technologies is almost unavoidable. Therefore, being aware about possible incompatibilities and issues makes migration project plan more accurate and feasible and consequently decrease risk of migration failure.

REFERENCES

[1] Bierhoff , K., Grechanik, M. and Liongosari, E. (2007). Architectural Mismatch in Service-Oriented Architectures. may. 4 {4. doi:10.1109/SDSOA.2007.2.

[2] Bishop, J. and Horspool, N. (2003). Cross-Platform Development: Software that Lasts. Annual IEEE/NASA Software Engineering Workshop SEW-30.39(2006), 9.

[3] Bitpipe (2010). Software Migration:Definition. doi :http://www.bitpipe.com/tlist/Software-Migration.html.

[4] Burgess, D. T. F. (2001). A General Introduction to the design of questionnaires for survey research. doi:http://www.leeds.ac.uk/iss/documentation/top/top2.pdf.

[5] codestyle.org (2008). Web Font Survey. Technical report. doi:http://www.codestyle.org/css/font-family/sampler-CombinedResults.shtml.

[6] Economides, N. and Katsamakas, E. (2006). Linux vs. Windows: A Comparison of Application and Platform Innovation Incentives for Open Source and Proprietary Software Platforms. In Bitzer, J. and Schroder, P. J. (Eds.) The Economics of Open Source Software Development. (pp. 207 { 218). Amsterdam: Elsevier. ISBN 978-0-44-452769-1. doi:DOI:10.1016/B978-044452769-1/50010-X. Retrievable at http://www.sciencedirect.com/science/article/B86TN-4PB7H82-F/2/21b1f996676f70a7d5fe58d0ee70cc87.

[7] Gauch, R. R. (2009). Measurements They ‘ever Exact. In It's Great! Oops, No It Isn't. (pp. 65{71). Springer Netherlands. ISBN 978-1-4020-8907-7. Retrievable at http://dx.doi.org/10.1007/978-1-4020-8907-7_8.

[8] Heymans, L., der Beken, T. V. and Wilson, B. (2007). Testing Techniques for the Cross-platform Migration of Very Large Interactive Applications. Software Maintenance and Reengineering, European Conferenceon.0,323-324.ISSN1534-5351. doi :http://doi.ieeecomputersociety.org/10.1109/CSMR.2007.46.

[9] Inglenet-Business-Solution (2001). A CASE STUDY OF PLATFORM MIGRATION FROM UNISYS 2200 TO UNIX ALBERTA BLUE CROSS PORT PROJECT. Technical report. Inglenet Business Solutions. doi:http: //www.inglenet.com/downloads/Blue Cross Case Study - Detailed.pdf.

[10] Karpov, A. and Ryzhkov, E. (2007). 20 issues of porting C++ code on the 64-bit platform. doi:http://www.viva64.com/content/articles/64-bit-development/ ?f=20 issues of porting C++ code on the 64-bit platform.html&lang=en&content=64-bit-development.

[11] Kharitonov, E. V. (2000). A Method of Making Subjective Measurements Compatible with Hierarchical Matrices of Preference Ratios. Measurement Techniques. 43, 747{751. ISSN 0543-1972. Retrievable at http://dx.doi.org/10.1023/A:1026689404649.

[12] Kuhn, M. (2009). UTF-8 and Unicode FAQ for Unix and Linux. doi:http://www.cl.cam.ac.uk/↑ mgk25/unicode.html.

[13] Matthias, R. D. F. M. N. E., Schonlau (2006). Conducting Research Surveys via E-mail and the Web. doi:http://www.rand.org/pubs/monograph reports/MR1480/index.html.

[14] McCarthy, S. P. (2008). Choosing the Right Platform for Trusted Cross-Platform Information Sharing. Technical report. doi:http://www.linux.com/learn/whitepapers/doc/11/raw.

[15] Mercer (2006). Migration Decision-Maker Interviews. Technical report. doi:http://download.microsoft.com/download/E/A/0/EA0F6F0B-BAA2-46B1-9EBC-7F28EFA7C508/MercerWhitePaper%20.pdf.

[16] MicrosoftTech (2006). UNIX Custom Application Migration Guide. (2nd ed.). Microsoft Tech Net.

[17] Nelson, M. (1995). C++ Program Guide to Standard Template Library. Foster City, CA, USA: IDG Books Worldwide, Inc. ISBN 1568843143.

[18] Oblitz, T. R. and Mueller, F. (2000). Combining Multi-Threading with Asynchronous Communication. In In Myrinet User Group Conference.

[19] Poniatowski, M. (2003). Linux on HP Integrity Servers. Upper Saddle River, NJ, USA: Prentice Hall PTR. ISBN 0131400002.

[20] Rajagopal, R. (1998). Windows NT, UNIX, NetWare migration and coexistence: a professional's guide. vol. 1. CRC Press.

[21] Ramanathan, B. F., R. (2004). Virtualization: Bringing Flexibility and New Capabilities to Computing Platforms. Technical report.

[22] Richter, K., Nichols, J., Gajos, K. and Se ah, A. (2006). The many faces of consistency in cross-platform design. In CHI '06: CHI '06 extended abstracts on Human factors in computing systems. New York, NY, USA: ACM. ISBN 1-59593-298-4, 1639{1642. doi:http://doi.acm.org/10.1145/1125451.1125751.

[23] Roth, M. (2009). Method for developing platform independent launchable applications.

[24] Springer (2001). Compatibility analysis. In Computer Science and Communications Dictionary. (pp. 259{259). Springer US. ISBN 978-1-4020-0613-5. Retrievable at http://dx.doi.org/10.1007/1-4020-0613-6_3222.

[25] SunMicosystem (2005). Converting 32bit Applications Into 64bit Applica-tions Things to Consider. doi:http://developers.sun.com/solaris/articles/ILP32toLP64Issues.html.

[26] Taleb, M., Se ah, A. and Abran, A. (2007). Patterns-Oriented Design Applied to Cross-Platform Web-based Interactive Systems. aug. 122 {127. doi:10.1109/IRI.2007.4296608.

[27] TechNet, M. (2006). Functional Comparison of UNIX and Windows. In Functional Comparison of UNIX and Windows. Microsoft Press.

[28] TIOBE (2010). TIOBE Index for January 2010. doi:http://www.tiobe.com/content/paperinfo/tpci/index.html.

[29] Weiss, G. J. (2009). The Great Virualization Delimma of Next Decade: What You Need to Know. doi:http://www.gartner.com/DisplayDocument?doc cd=164938&ref=g BETAnoreg.

[30] Wilson, B. and Beken, T. V. d. (2003). Observations on automation in cross-platform migration. doi:http://soft.vub.ac.be/FFSE/Workshops/ELISA-submissions/08-Wilson-position.pdf.

[31] Wojtczyk, M. and Knoll, A. (2008). A Cross Platform Development Work flow for C/C++ Applications.

[32] Xu, L., Xu, B., Nie, C., Chen, H. and Yang, H. (2003). A Browser Compatibility Testing Method Based on Combinatorial Testing. In Lovelle, J., Rodrguez, B.,Gayo, J., del Puerto Paule Ruiz, M. and Aguilar, L. (Eds.) Web Engineering.(pp. 310{313). Lecture Notes in Computer Science, vol. 2722. Springer Berlin /Heidelberg. Retrievable at http://dx.doi.org/10.1007/3-540-45068-8_60.