International Accessibility of Open Source Software

Yu Wang   E. James Whitehead, Jr.
Department of Computer Engineering
University of California, Santa Cruz
Santa Cruz, CA 95064
{ywang,ejw}@soe.ucsc.edu

July 2001

Abstract

Though open source sofware has achieved tremendous success in recent years, no existing literature examines issues of accessibility for international users. In this paper we present an international accessibility model which involves adoption of open source software and participation in the development community of software by international users. Based on that model, we first classify open source software roughly into three categories based on its target user groups, i.e., system administrators, developers and end-users, and discuss a list of desired accessibility properties from the viewpoints of international users. Our investigation of some high-profile open source projects shows that technologically difficult problems are solved rather elegantly in some projects while seemingly easy problems such as translation of manuals are ignored or poorly addressed. We explain possible factors that can lead to the differences among these projects and conclude that most existing open source software still lacks desired accessibility properties for international users, and various contributing factors such as cultivation of online user communities, support from commercial companies, educational and propagandistic roles of press, localization of documentation, highly modular design and building of integrated Web environments for global developers and users need to work together to improve the status quo.

1 Introduction

Open source software development has received increasing attention in recent years due to the success of high-profile projects such as Linux, Apache, and Sendmail, that have achieved greater success than equivalent closed-source commercial products. For example, Apache handles more than 60% of the publicly visible websites according to Netcraft's Web server survey [1] and Sendmail is asserted to handle over 80% of the world's email traffic [2]. However, as most of the well known open source projects are based in the United States, or an English-speaking country, open source development is currently biased towards English speakers.

Open source projects leverage the Internet, using it as a distribution and development medium. Due to the worldwide deployment of the Internet, the open source movement has the *potential* to be truly international, spreading its ideals of open access to source code, and collaborative development by technical meritocracy, to hundreds of millions of people worldwide. However, since many people do not understand English, the English focus of the documentation and user interfaces of open source software reduces the impact of the open source movement.

For open source to meet its full potential as a *global* movement, the international accessibility of open source software must be improved. A close analysis of the Netcraft Web server survey illustrates this point. Over all domains, the ratio of the number of sites running Apache to those running Microsoft's Internet Information Services (IIS) server is 3:1 [1]. Even though the Apache server has relatively good international accessibility, with existing translations of Apache books available in multiple languages, in non-English speaking countries, the ratio drops to approximately 1:1, with some countries reporting slightly more Apache installations (the July 2001 report reports Malaysia with 7457 Apache to 6717 IIS), and others showing slightly more IIS (e.g., Singapore reports 7574 IIS to 6895 Apache). We attribute this difference primarily to the superior international accessibility of IIS.

In fact, despite the difficulty of adapting software to the needs of specific user/language groups, a process called *internationalization* (i18n) and *localization* (l10n), several U.S. based computer companies such as Digital (now a division of Compaq), HP, IBM, Microsoft and Sun are quite successful at it, reaping significant rewards as a result. For these companies, upwards of a half of their revenues were derived from international sales between 1991 and 1995 [3]. More recent figures from U.S. Securities and Exchange Commission confirm that this is still true [4]. For example, in their reports, HP [5], IBM [6] and Sun [7] stress the importance of their world-wide operations and their efforts to attract international customers. They have achieved this result by focusing on increasing the accessibility of their software for non-English speaking customers.

As the open source movement expands beyond its traditional user base of technically sophisticated users, documentation becomes increasingly important. Open source developers and advocates must take international accessibility seriously and regard documentation as an integral and equal part of an open source software product. Poor documentation or lengthy documentation turn many potential users away, thus reducing the impact of open source software.

Our survey of the available literature on open source reveals that, despite its importance, international accessibility has not yet been investigated. In this paper we provide a model for international accessibility in open source software (Section 2). This model details the steps performed in acquiring, installing, and operating an open source program, highlighting the importance of documentation in this process. Successfully using a program is a prerequisite for joining its development team, and we detail aspects of international accessibility in this process in Section 2.4.

In Section 3, we classify software into three categories according to its target user groups and then describe different levels of internationalization and localization requirements from the viewpoints of these user groups as well as some other factors that can influence international accessibility. Then we present the investigation results of some open source projects which exhibit different levels of accessibility to international users. In Section 4, we analyze the major factors that lead to the difference among these open source projects. Section 5 concludes this paper.

2 A Model for Accessibility of Open Source Projects

What if you were a computer user in the People's Republic of China who wanted to use an open source application. You would first need to learn enough about that program to know that it would, in fact, meet your needs. Assuming it does, you will need to download the application, or acquire it on CD. If the application only comes with source code, you need to compile it. Once the binary is available, it needs to be installed. With the installation completed, you can finally start addressing the operation of the software, including use of configuration files, interacting with the user interface, and so on. All of these tasks, from assessing its capabilities, to installation and operation, require reading and understanding often complex instructions and documentation. Since the preponderance of information available about open source projects today is written in English, installing and using open source applications can be a daunting task. As a Chinese computer user, you may have some understanding of English, and thus be able to laboriously learn about and use open source software. However, many Chinese computer users do not understand English, and for them, the richness and diversity of open source applications are sealed off behind an impenetrable wall of language.

A significant benefit of open source software is the ability to participate in the development of an application. It is extremely unlikely that someone will contribute to an open source project without actually using the application being developed, so issues of installation and use must be addressed before someone can even think about contributing source code. Once they do, further issues arise. Source code is typically commented only in one language, and the discussion list may only have single-language messages. Error reports may only be in one language. If the language of the project is not the contributor's native language, it becomes more difficult to participate.

Our model of the international accessibility of open source projects involves two primary activities: adoption of open source applications and participation as a developer in an open source development team. Adoption is a prerequisite for participation as a developer, and includes learning about, installing, and operating open source software. Participation in a project involves access to mailing lists, bug tracking information, and ability to use the project's development environment. These issues are described in depth below.

2.1 Learning About the Software

Motivations for adopting open source technology vary. Some people will have a specific set of goals that they seek to have fulfilled by an application, such as the desire to serve Web pages. Others may be technology dilettantes, interested in exploring a wide range of applications for the joy of learning. In either case, since there are often many application choices in each application domain (e.g., multiple HTTP servers, word processors, graphics programs, etc.) and only a finite amount of time to explore them, the first step in adoption of open source technology is reading available materials to determine if the technology meets intended goals. In this step, the potential user seeks to bridge the knowledge gap between the rich, often technology-focused understanding the development team possesses about an application, and their own limited understanding of the technology. Potential users need more than just a list of features; ideally fundamental conceptual models (such as the client-proxy-server architecture of the Web for an HTTP server) will be conveyed as well.

Additionally, potential users need an understanding of the runtime environment of the software. This includes software issues such as supported operating systems, dependencies on other software libraries, and security implications, as well as such hardware issues as supported processors, processing speed, memory needs, and additional hardware required. Third-party software libraries can have a significant negative effect on accessibility if it is not integrated into the installation process, since each library requires going to another Web site, and starting the entire adoption process anew.

For many open source projects, the official website for the software is the only information source available to potential users, and hence its educational role is large. It is the first place that users come to know the software, especially when the software is not well publicized. It is also the place where users can acquire a software package with its accompanying documentation. Clearly, accessibility is increased if information about a project is available in multiple languages, since each additional language increases the pool of potential users.

Open source portal sites such as Freshmeat [8], are important intermediaries in the information seeking process, providing a condensed and filtered view of open source project activity. Once project information is available in multiple languages, accessibility improvements in these sites provide a second-order effect. When intermediary websites support multiple languages, it improves international accessibility.

Word of mouth is another effective mechanism for disseminating usage information and cultivating both potential users and developers. Online user communities such as mailing lists and Web-based discussion forums, as well as physical world conferences and users group meetings are all places where people can exchange information. When only English-based communities are present, project accessibility is reduced.

Several difficulties surround the establishment of non-English speaking user communities. First, there is the bootstrap problem of finding an individual willing to act as a project champion in a specific language. Resource issues come into play, such as where to host a mailing list, and a Web site for the project. Culture can also have an effect. For example, in Malaysia, commercial closed-source software development preceded open source software development and is well established [9]. Since it is a local tradition not to challenge established practice, this tradition has acted to slow adoption of open source technology in Malaysia.

The absence of an international user community for a technology can lead to disconnects between potential users and the project developers. Since potential users do not know where, who, and how to request new features, developers may mistakenly assume that users have no i18n/l10n requirement and do not realize that they may need to take the initiatives to make their software more accessible to international users.

Widely used open source projects can have enough users in a given language to make a press run economically feasible. In this case, for-profit publishers are valuable intermediaries distributing information about open source projects in multiple languages. Printed books and manuals are especially appealing when users cannot view the documents online, for example, during the process of installing certain system software. O'Reilly is a well known example, publishing translated books on open source software as well as internationalized information processing [10, 11]. This helps to educate non-English speaking users on how use the software and potential developers on how to internationalize and localize their software.

2.2 Installing Open Source Software

After learning about an open source application, a potential user may then decide to install the software on their own computer system. This is a potentially tricky process of fitting the configuration of a user's system within the range of system configurations considered during the development process. In the worst case, installing an application can cause system configuration changes that cause pre-existing applications to stop working, or work incorrectly. In the case of server software, security holes may be introduced. Poorly crafted code could cause loss of data.

Due to the risks of introducing new software into an existing system, it is crucial that users understand the impact of installing a piece of software. Potential users perform a risk/reward analysis for their particular use context where they assess the benefits, which they gathered during the process of learning about the software, against the configuration and security risks of installing and running the application.

To install a software system, it is necessary to provide information concerning the mechanical aspects of acquiring the application. For many open source projects, the primary means of distribution is the Internet, and so the software is downloaded as either source code, or a precompiled binary. Long-running, well-known projects may be available on CD compilations of open source projects, or in the distributions of the Linux operating system. If the software is downloaded as source code, a process of compilation is required, involving customization of makefiles and source code for the user's system. Instructions for this operation can typically be found in a "README" file. While tools such as *autoconf* have standardized and simplified the process of customizing source code for a specific platform, advanced uses of these tools still occur (e.g., Apache's use of configuration options to select which modules to precompile and install), and hence documentation concerning the customization and compilation process is often necessary. Certainly, the more this documentation is available in multiple languages, the easier it will be for people speaking these languages natively to compile a open source project.

Even with the use of autoconfiguration technology, errors in the compilation process still occur. In this case, the person installing the application is left with the choice of either reading and deciphering the problem in the source code (requiring significant technical sophistication), or directly contacting the user community for the software by making a post to a mailing list offering technical support.

Once the executable binary has been created, it must be installed and configured. The installation process can range from the trivial to the complex, since many applications can be run in-place without any further installation, while others require the creation of directory trees populated with specific configuration and documentation files. While automated support for this operation is typically available, it is helpful to know ahead of time what consequences attend the installation process. Once installed, many projects have configuration files whose syntax is often obscure (and unforgiving) and where small changes can have significant impact on security and performance.

Following the same pattern as for initially learning about an open source application, both the project web site and third-party intermediaries play a role in disseminating information about project installation and configuration. The project's Web site is the most authoritative, and many installations begin and end here. Users groups can help by running tutorials on how to acquire and install a project, and books about open source projects typically include installation instructions. Having this information available in multiple languages increases the international accessibility of the project.

2.3 Operating Open Source Software

Once the software is successfully installed, issues of operating the software come to the surface. Operating instructions for the software need to be available in the native language of the user, and the user interface for the software needs to present menus and dialogs in the user's native language, as well as permit native language input.

2.3.1 Native Language Documentation

In the best case, documentation should consider the cultural background of the target user groups. For example, the literal translation of manuals is far from sufficient due to the the difference in learning habits across cultures. For example, a U.S. or German user needs only minimal information to operate comfortably and successfully, while a Japanese or French user may need much more contextual information to operate at the same confidence level [12]. Thus a quick tutorial and then an itemized manual may not be the most suitable form of documentation for some user groups.

Another example is to avoid audible feedback such as sounding a bell for mistakes. When localizing software for Japanese users (*Japanization*), especially if the software is to be used in workplace, a Japanese user may feel embarrassed if his colleagues hear frequent bell rings and know that he is making mistakes. "Saving face" is important in Japanese culture and should be considered in Japanization [13]. This highlights the importance of drawing upon research results from other disciplines, such as psychology and anthropology, to make software and its documentation more suitable for its users.

Third parties play an important role in the process of informing users how to operate open source applications. Especially in the case of programming languages and server software, face-to-face tutorials and online discussion lists are invaluable resources. Commercial publications describing the operation of open source projects can often provide better quality documentation than the project's Web site, and due to the profit motive, often are translated into many languages.

2.3.2 Native Language User Interface

Software i18n/l10n has two levels, surface and cultural [12]. The surface level deals with character sets and encodings, text handling, date/time formats, currency, and so on, while the cultural level tries to fit the software in the user's cultural background, such as color preferences and learning habits. We expand on character sets and text processing below.

Character Sets and Encodings. To produce a user interface in a user's native language, a developer must use character sets and encodings. Different user/language groups use different characters. The English alphabet, Arabic numbers and some other symbols are the most frequently used characters by English-speaking users. These characters form a *character set* and the representation of a character set in computers is called *character encoding*. ASCII is a well-known encoding method that uses 7 bits to represent a character. In fact it also specifies a character set at the same time, as it includes English alphabet, numbers, U.S. currency and some control characters that are commonly used in computers. Another example is ISO8859-1 [14], an extended character set for the U.S. and Western European countries which uses 8 bits to represent one character. As ISO8859-1 uses the highest bit of one byte as well, a software developer should ensure that his code is 8-bit clean, i.e., it can handle 8-bit characters transparently without, say, stripping away the highest bit.

For some other countries, notably China, Japan and Korea,[1] their languages have large sets of characters. Commonly used characters number in the thousands. If seldom used characters are included, the number rises to more than ten thousand. Two or more bytes are needed to represent these characters, and several character encodings (i.e., UTF-8, UTF-16, UCS-2, UCS-4) exist for expressing these character representations as sequences of bytes [10].

Initially, each language group had one or more idiosyncratic encoding schemes, and supporting multiple encodings within one program was difficult. Even when support for multiple encodings was not required, it was still tedious to rewrite a program for each a different character set. The Unicode [15] standard addresses these problems by providing a uniform encoding of all currently used characters, as well as many ancient characters, in one universal character set. Though Unicode provides a forward migration path, current practice still involves support of both Unicode and locale-specific character sets (a.k.a. *legacy* character sets) used in the target market in the software. Localized Microsoft Windows 98 is a typical example. It provides multibyte handling at system level and conversion between native encodings and Unicode is on the fly whenever necessary. This process is quite seamless and users are unaware the conversion is occurring [10].

Once it is possible to express characters in multiple languages, word expansion in elements of the user interface also must be considered. A somewhat extreme example is that the German word for "preference" is "bildschirmeinstellungen." Therefore, a developer should consider these factors in budgeting space for user interface elements [16].

Text Processing. The requirements for text processing differ across languages. Although the writing direction of many languages is from left to right, Arabic and Hebrew for example are written from right to left. Software developed for these languages is required to provide bi-directional support.

In writing systems such as Chinese, a character (or an ideograph) is of square shape, thus there are no concepts of proportional spacing, ascent or descent that are some common properties of Latin-based fonts. This presents new problems for typeface design. Additionally, character entry is also a problem for users of a large set of characters as no commodity keyboard can be built with each key corresponding to each character. Since character display and entry are usually handled at the system level, application developers can safely ignore such issues.

Sorting of characters is also affected by different writing systems. For example, Chinese characters can be sorted according to pronunciation, primary radicals (the radical that carries the meaning of a character) or the number of strokes of characters. Case sensitivity is not applicable to such writing systems. The hyphenation rules and justification of texts also vary across writing systems. For example, Japanese has *rubi* characters, which are *kana* characters (one part of Japanese scripts) written above a *kanji* character (another part of Japanese scripts) to annotate the pronunciation of the kanji character [11].

A *locale* is the runtime language environment of a program; well-designed software adapts its behavior according to the user's locale. A locale specifies the language, territory (for example, English either in UK or in US) and

[1]Due the cultural affinities among these countries, they are often cited together as CJK.

character set (a language can have several character sets). The benefit of locale is that a bilingual user can switch his environment easily.

In closed-source development, best current practice is a blend of run-time and link-time approaches to internationalize and localize software [3]. Run-time approaches make use of external message and resource files to change a program's behavior, while link-time approach makes use of dynamic libraries which may include locale-dependent routines for information presentation and handling. Localization is usually performed in-house. Game companies sometimes out-source, relying on local companies in each target market to do the localization work, especially when they are not familiar with the cultural background of potential game players. Another interesting approach is used by a Norwegian software company [17]. They integrate professional freelance translators from target countries in the development process and expect to obtain higher product quality and greater flexibility at lower costs.

2.4 Participation as a Developer in an Open Source Project

The following factors can influence developers' participation in the development of open source software projects.

2.4.1 Modular Design

A crisp separation of concerns into modules is an important design technique for any software project. International accessibility of open source projects can be increased by isolating language-dependent code into one or more clearly identified modules. This allows localization projects to proceed independent of the development of the language-independent functionality allowing even non-programmers to contribute to the localization process. The Mozilla Web browser project [18] provides an good example of this, since it separates the language independent and language specific elements, providing the additional feature of language-packs for each specific language.

2.4.2 Internet-based Collaboration and Distribution

Mailing lists are the lifeblood of any open source development effort, carrying the majority of communication among project members. While ideally a mailing list would allow discussion to take place in multiple languages, thus easing participation for people from multiple linguistic backgrounds, the present state of automatic translation software does not permit this. Instead, project members pick a language, typically English, and solely use that for their discussion.

Web-based development environments, such as those offered by SourceForge [19], provide access to source code, Web-based bug tracking, and mailing list archives. These Web sites have a user-interface, and accessibility can be improved, at least slightly, by internationalizing it. At present, SourceForge can present its first level webpages in more than 10 languages. However, this is unsatisfactory as users still need a working knowledge of English to understand the mail messages and source code comments once they have navigated to them.

3 Accessibility in Open Source Software

Just how accessible are existing open source projects? We explore this question by examining nine well known open source projects. We roughly classify open source software into three typical categories according to the software's target user group: software for system administrators, for developers, and for end-users. In each category we examine three projects to see how well they are accessible to international users.

These projects include:

- Software for system administrators, such as Web server and email server.

  Apache [20] An HTTP server.

  Sendmail [2] A widely used mail server.

  Qmail [21] A modern mail server designed to provide a more secure alternative to Sendmail.

- Software for developers, such as database management and development system, programming languages and their support libraries.

  MySQL [22] One of the most popular Linux databases. It also runs on a broad range of other platforms.

Perl [23] A popular scripting language.

Tcl [24] Another popular programming language and it is backed by Scriptics, Inc. Its associated graphical toolkit, Tk, makes it easier to write cross-platform graphical programs.

- Software for end-users, such as text editors and Web browsers.

GNU Emacs [25] A widely used text editor, ported to many platforms.

Mozilla [18] A Web browser based on Netscape Communicator 5.0, developed by the open source software community with Netscape's support and coordination.

KDE [26] A graphical desktop environment for Unix and Unix-like workstations.

Software that belongs to different categories targets different users with different expertise, which can lead to different accessibility requirements as follows:

Packaging properties Does the software provide localized installer and localized documentation? Software having these properties sometimes is much more important than the functionality of the software itself in terms of adoption by non-English speaking users.

Code properties These properties are usually required for the software to run properly or for users to understand the run-time behavior of the software. Important code properties include support of multibyte characters and localizable/localized user interface (UI).

Language-specific processing properties These properties are required either for developers or for end users.

For developers, the software should provide necessary facilities to develop locale dependent applications such as sorting of multibyte characters according to different criteria. For end users, the requirements for locale-dependent information processing are application and language specific. For example, text editors should handle insertion and deletion of multibyte characters correctly.

Commercial support for non-English speaking users It can include books from publishers, translated manuals from commercial vendors distributing open source software and technical support from commercial companies.

Non-English speaking user community It can include mailing list, Web based discussion forums and etc. Though it can usually be reasoned out that if there is commercial support for non-English speaking users, there must exist non-English speaking user community. Here we look for the online user communities that are freely available to interested users who want to communicate in their own languages, as an often tauted merit of open source software is that users may get better support than what is offered to closed-source software by commercial companies.

Based on above discussions, it seems that the server side software requires the least i18n/l10n efforts. Since they usually implement culture-neutral functionality such as transferring files and emails, i18n/l10n is rather a straightforward process. Translation of documentation usually is enough. Translation of output messages is also easily possible, provided they are not hard-coded in the program.

As to the developer-side software, it requires much more effort. Not only is the above translation needed, but also some locale-dependent or internationalized functionality should be added. For example, support for Unicode and/or some "legacy" encodings and locale-dependent functionality (sorting, etc.) are desired for a DBMS. Such facilities can help programmers to develop locale-dependent applications without the need to reinvent the wheel.

As to the software for end-users who may be casual and nontechnical, the requirements are much more stringent. In addition to translation of documentation and some locale-dependent functionality, the whole user interface should be localized to accommodate the users' requirements. If more locale-dependent functionality is required, more efforts for i18n/l10n are needed.

However, our investigation results of these open source projects are somewhat unexpected. They are briefly summarized in Table 1 and detailed discussions are as follows.

	Localized installer	Localized documents	Localized UI	Unicode support	Legacy character set support	Non-English speaking user community
Apache	No.	Yes.	No.	N/A	N/A	Yes.
Sendmail	No.	Yes.	No.	N/A	N/A	Yes.
Qmail	No.	Yes.	No.	N/A	N/A	Yes.
MySQL	No.	Yes.	Yes.	Ongoing work.	Yes.	Yes.
Perl	No.	Yes.	No.	Yes.	Yes.	Yes.
Tcl/Tk	No.	Yes.	No.	Yes.	Yes.	Yes.
GNU Emacs	No.	Yes.	No.	Yes.	Yes.	Yes.
Mozilla	No.	Yes.	Yes.	Yes.	Yes.	Yes.
KDE	No.	Yes.	Yes.	Yes.	Yes.	Yes.

Table 1: International accessibility related properties in some open source software

3.1 Localized Installer

From Table 1, we can see that Apache, Sendmail and Qmail do not provide localized installers. Besides, installation from source is strongly recommended, especially when applying security patches. Here in server side software, security and performance are much more important than user convenience and installation of this kind of software requires much more expertise.

MySQL, Perl, Tcl/Tk, GNU Emacs and KDE do not provide localized installers as well. However, installation from source is not necessary and software update is less frequent than server side software. In addition, as they are usually bundled in major Linux distributions, installation seems easy for Linux users. In comparison, Unix users interested in using them are not so lucky and they have to install it by themselves or seek help from system administrators.

Though Mozilla Web browser does not provide any localized installer, it does provide a graphical installer which seems to be much easier for people to go through the installation process. Installation of additional language packs can also be done easily with pull-down menus.

3.2 Localized Documents

Here localized documents include both localized manuals, HOWTOs and FAQs that accompany the software and books or other printed materials by either publishing houses or software vendors. We do not consider other documents that are scattered in the Web such as mailing list archives.

We find that O'Reilly and Associates Inc. has published many books on Apache, Sendmail, MySQL, Perl, Tcl/Tk and GNU Emacs in a variety of languages. This may be due to the fact that they are really big and successful open source projects and third-party writers are willing to translate these books for non-English speaking users.

As to the translation of manuals and project related documents, the results are quite different among these projects and they are summarized as follows:

- For Apache, at least Japanese and Russian Apache User Groups have been working on translation of the Apache manuals and have set up their own websites. However their efforts are not shown on the official Apache website. It would be nice if the official Apache website can have more coordination with these local user groups and provide information about them.

  At the time of writing, the Apache documentation project [27] spawned in July of 2000 is still largely confined to improvement on Apache's documentation for English-speaking users.

  It seems that, due to limited resources and committed efforts from international translators, Apache still lags behind in its document localization compared with Microsoft IIS Server. Therefore, it fails to compete in some markets where non-English speaking users dominate.

- For Sendmail, Perl, Tcl/Tk and GNU Emacs, we are not aware of any ongoing translation projects of their documentation at the time of writing.
- Since Qmail is much simpler than Sendmail, its documentation has been translated into several languages. From an international user's perspective, Qmail is much more attractive than Sendmail in that it is secure and easy to use.
- MySQL does have some contributed translated manuals in several languages such as Chinese, French and Japanese, however they are not bundled with the distribution.
- Mozilla offers many language packs. Once they are installed, users can get online help in their own languages. There have been about 50 registered localization projects for Mozilla [28] and most of them have provided languages packs for different milestone versions of Mozilla.
- KDE website provides links [29] to websites which are devoted to users speaking other than English and links [30] to GUI and documentation translation projects being conducted in about 16 languages. All these projects' status can be conveniently checked on their websites.

3.3 Localized User Interface (UI)

For command line based software, localized user interface means that the output messages of the running software can be customized to other languages. For software with graphical user interface (GUI), we are interested in whether the GUI can be localized. We find that only a few of the software products investigated provide such feature, which are discussed as follows.

- In MySQL, error messages can be customized to user's own language.
- Mozilla provides extensible locale support, i.e., the additional locale system can be "dropped in" without modifying the base binary. From the viewpoint of an end-user, he can just download the base binary and the additional languages packs that he needs. Installation of languages packs is very convenient and a user can switch among different languages with ease. All GUI elements can be localized.
- KDE also has localized GUI which has been discussed in Section 3.2.

3.4 Legacy Character Sets and Unicode Support

For the service side software, it is enough to be 8-bit clean. Since Apache, Sendmail and Qmail are all 8-bit clean, we do not discuss them further thereafter.

Our findings of other software are summarized as follows:

- MySQL supports different character sets that can be specified at compile and run time. However sorting of characters in different character sets is not well documented in manuals and users may need to dig into the source code to make sure how the sorting is done as sorting functions are very important in a DBMS. Since some users may not get enough information from documentation and source code, they may feel uncomfortable when they need to make use of the not well documented features (as discussed in Section 2.3.1) and they may resort to other competing products that can offer better documentation.

  There is currently ongoing work on "hacking" MySQL code to add Unicode support, so that a MySQL database can store characters from multiple languages. Basis Technology Inc. [31] has worked out a plan of three phases to enable Unicode support in MySQL. From discussions in MySQL's mailing list, we know that the company is willing to contribute its code and cooperate with other MySQL developers to add full Unicode support to MySQL.
- Perl supports many legacy character sets via locale mechanism. Perl's support for Unicode is still evolving at the time of writing.

  Unlike MySQL, the Unicode and locale support and the use of them are well documented in the manuals. Additionally, the latest edition of "Programming Perl" [32], one of the definitive books on Perl, also addresses

Unicode and its use, which reflects the committed efforts to internationalize Perl from Perl's development community.

Another related project called Perl Wrappers for ICU Project (PICU) [33] is going on, whose objective is to build a Perl Extension Subroutine wrapper around the International Components for Unicode (ICU), which itself is an IBM-backed open source project to develop a C and C++ library that provides robust and full-featured Unicode support on a wide variety of platforms [34].

- Tcl/Tk supports many legacy character sets and Unicode and their use is also documented in a definitive book on Tcl/Tk programming [35] as well as its manuals.
- Although it is not obvious from the information provided in the GNU website, GNU Emacs does have a multilingual text processing environment. The multilingual environment is based on the Mule (Multilingual enhancement to GNU Emacs) project that was started in Japan [36]. Its goal is to provide multilingual text processing environment in Emacs so that a user can edit multiple character sets, say ASCII, Japanese, Chinese, Hebrew, etc. in one buffer. It has been incorporated into the standard distribution of GNU Emacs since version 20.

  GNU Emacs' support for multilingual text processing is truly excellent. Handa et al. [37] provide an excellent exposition of the unified and extensible mechanisms underlying the design of Mule.
- Mozilla and KDE support many legacy character sets and Unicode.

3.5 Commercial Support

If we do not consider publishing translated books and manuals as a form of commercial support, then the commercial support of some of these software products for non-English speaking users is not so obvious to us.[2] However, we do find some information about the following software:

- Sendmail Inc. [38] is a commercial company which strives to develop better user interface for configuring Sendmail. They target clients in the U.S. and some European countries and their website can be viewed in English, French and German.
- For MySQL, there are already some consulting companies serving local users in their countries. For example, SoftAgency [39] is a Japan-based company serving Japanese users and it has links to local Japanese user groups and localized Japanese manuals for MySQL.
- For Qmail, there have also been quite a few companies offering commercial support for European users [40].

3.6 Non-English Speaking User Community

Since O'Reilly and Associates Inc. have published books on Apache, Sendmail, MySQL, Perl, Tcl/Tk and GNU Emacs in different languages, it can be reasoned out that there exist non-English speaking user communities of these software products. The availability of commercial support for Sendmail, MySQL and Qmail also demonstrates the existence of non-English speaking user communities. We have also found some other evidences for the following software:

- Apache: Japanese and Russian websites exist.
- Qmail: Japanese, Korean, Russian and Spanish websites exist.
- MySQL: German, Japanese, Korean and Portuguese mailing lists exist.
- Perl: There are established Perl Groups around the world [41] and there is also an annual Perl workshop in Germany [42].
- Mozilla and KDE: They have many language specific websites and many localization projects are going on.

[2]Mozilla and KDE seem not in great demand for commercial support as the former is as easy to use as Netscape Communicator and the latter is usually bundled in installation and is also very easy to use.

3.7 Other Aspects

In the software we have investigated, the configuration files are written in English, which presents another problem for non-English speaking users. However, this problem is not insurmountable provided a good user interface can hide much, if not all, of the English words and phrases used in configuration files. For example, administrators of localized Microsoft IIS and Exchange Server can get started much easier even if they do not know much about English.

4 Discussions of Investigation Results

Just as what we have discussed, technically speaking, translation of documentation and localizing GUI elements are relatively easy while adding locale-dependent functionality such as editing multibyte characters is relatively difficult. Thus we had expected to observe that the former was done better than the latter in open source projects. However, our findings have shown that the reverse is true. It seems that most of the technologically difficult problems are solved rather elegantly in some open source projects such as the multilingual text processing in GNU Emacs and the support for a variety of character sets in MySQL, Mozilla and KDE. The translation of documentation lags far behind. Maybe it is what many open source advocates have argued, that open source software tends to be more or less geared towards technically sophisticated users [43], or more accurately, those users who are also comfortable with an English-based environment.

Except Mozilla and KDE as outstanding counterexamples, most of the open source projects still lack good translated documentation and a localized user interface for non-English speaking users. Following, we analyze two important factors that can lead to these differences in the efforts to make software accessible to international users.

4.1 Originators

The first factor is whether the originators of open source projects have the foresight of i18n/l10n or consider it only as an afterthought. Past experiences have shown that it is rather difficult and cumbersome to internationalize/localize software if it is not considered in the original design. For example, it is stated [35] that for Tcl/Tk, the internal changes required to support Unicode caused a major overhaul that touched nearly the entire implementation. As we have shown earlier, the design of GNU Emacs is somewhat ethnocentric and does not seem to take non-English speaking users into account in the first place. For example, many operations of GNU Emacs have mnemonics that are only meaningful to English speaking users. Even though GNU Emacs is enhanced with multilingual text processing capabilities, the requirement to have a working knowledge of English will hinder its adoption by non-English speaking users. Contrary to this, Mozilla exhibits a well designed architecture for i18n/l10n. Perhaps this is partly due to the fact that the developers in Netscape have had i18n/l10n experiences in earlier versions of Netscape Navigator and Communicator and they have been targeting global market, so they can bring a clearly defined vision into the development of Mozilla.

4.2 User Communities

The second factor is related to the development of the open source community in other non-English speaking countries. Stodte [9] has given a brief description of the local open source communities outside the U.S., which are quite unbalanced due the the different resources available and the different cultural environments they work under. Since the originators of open source projects may lack the necessary cross cultural background to localize an open source software product, the best they can do is to follow some rules so that later i18n/l10n may be done more easily. For example, they can write 8-bit clean code, not to hard code messages in programs and not to assume that all characters are one-byte long. The actual l10n or m17n usually needs to be done by local user communities, so the development of local user communities has a major influence on software l10n/m17n.

For example, Japanese user communities are quite active in modifying or localizing U.S. initiated software to meet their need for Japanese information processing. They are quite successful as they usually enjoy abundant resources for development and a large English-speaking user base. For example, the Multilingualization Organization [44] is supported by the Japanese Ministry of International Trade and Industry and is active in several m17n projects such as Mule and Tamago, a CJK character input system. The efforts from Japanese and other non-native English speaking users help to add locale-dependent functionality to some open source projects. However, since most of them have at least working knowledge of English, they usually are not strongly motivated to push the documentation and user interface to the same quality as the functionality of the software itself. For example, the UI of GNU Emacs remains English-based.

On the other hand, it is much more difficult for open source software to penetrate some countries where resources are scarce and a broad English-speaking user base is not possible. Localization efforts from these communities are relatively marginal, despite the fact that the efforts from them are needed most. This is especially true in some developing countries where English is not widely spoken. Poor communication due to scarce resources together with insufficient English knowledge reduces the number of potential contributors to open source localization efforts in these countries.

From the two factors discussed above, we conclude that: English is still the predominant language used by open source developers even when their native language is not necessarily English. Open source developers and technically sophisticated users are still much more concerned about the functionality of the software, thus the technically difficult problems of character encodings, international text processing are solved quite elegantly while other accessibility problems are largely ignored. Notable exceptions are the devoted efforts shown in the i18n/l10n of Mozilla and KDE as well as the many translated books published by O'Reilly who itself is an active advocate of open source software.

5 Conclusion

We have presented a model for international accessibility in open source software, which involves both adoption of the software and participation in the development by international users. Based on that model, we investigate some existing high-profile open source projects. The results show that open source project software exhibits different levels of accessibility to international users and developers. Except Mozilla and KDE as counterexamples, most open source projects we have investigated exhibit the idiosyncrasy of elegantly solving technologically difficult i18n/l10n problems while the seemingly easy task of translating documentation is done poorly. Then we explain the possible reasons that are related to the originators of open source projects and the development of user communities in non-English speaking countries. We argue that without various contributing forces working together, such as cultivation of online user communities, support from commercial companies, educational and propagandistic roles of press, localization of documentation, highly modular design and building of integrated Web environments for global developers and users, it is questionable whether open source software can receive global support, especially from those non-English speaking user communities.

References

[1] "Netcraft Web Server Survey." `http://www.netcraft.com/survey/Reports/`, last checked 07/16/2001.

[2] "The Sendmail Consortium." `http://www.sendmail.org`, last checked 07/16/2001.

[3] P. A. V. Hall and R. Hudson, eds., *Software without Frontiers*. New York, NY: John Wiley and Sons, 1997.

[4] "U.S. Securities and Exchange Commission." `http://www.sec.gov`, last checked 07/16/2001.

[5] "U.S. Securities and Exchange Commission Form 10-K Filed for Hewlett-Packard Company." `http://www.sec.gov/Archives/edgar/data/47217/0000912057-00-002692.txt`, last checked 07/16/2001.

[6] "U.S. Securities and Exchange Commission Form 10-K Filed for International Business Machines Corp." `http://www.sec.gov/Archives/edgar/data/51143/0000912057-00-011206.txt`, last checked 07/16/2001.

[7] "U.S. Securities and Exchange Commission Form 10-K Filed for Sun Microsystems Inc." `http://www.sec.gov/Archives/edgar/data/709519/0000891618-00-004740.txt`, last checked 07/16/2001.

[8] "Freshmeat." `http://www.freshmeat.net/`, last checked 07/16/2001.

[9] M. Stodte, “Open Source around the World – How the Open Source Community Operates outside the United States.” `http://www-4.ibm.com/software/developer/library/worldos.html`, last checked 07/16/2001.

[10] K. Lunde, *CJKV Information Processing: Chinese, Japanese, Korean and Vietnamese Computing*. Sebastopol, CA: O’Reilly and Associates, Inc., 1999.

[11] K. Lunde, *Understanding Japanese Information Processing*. Sebastopol, CA: O’Reilly and Associates, Inc., 1993.

[12] W. Gribbons, “Designing for the Global Community,” in *Proc. of IEEE Intl. Professional Communication Conf.*, (Salt Lake City, US), pp. 261–273, 1997.

[13] P. Russo and S. Boor, “How Fluent is Your Interface?: Designing for International Users,” in *Proc. of ACM on Human Factors in Computing Systems*, pp. 342–347, 1993.

[14] “ISO 8859 Character Sets – Latin 1.” `http://czyborra.com/charsets/iso8859.html#ISO-8859-1`, last checked 07/16/2001.

[15] “The Unicode Consortium.” `http://www.unicode.org`, last checked 07/16/2001.

[16] N. Kano, *Developing International Software for Windows 95 and Windows NT*. Redmond, WA: Microsoft Press, 1995. `http://msdn.microsoft.com/library/default.asp?URL=/library/books/devintl/S24AB.HTM`, last checked 07/16/2001.

[17] H. Wigestrand, “Innovative and Interactive Internationalization of Software and Documentation,” in *Proc. of IEEE Intl. Professional Communication Conf.*, (Québec City, Canada), pp. 141–144, 1998.

[18] “The Mozilla Project.” `http://www.mozilla.org`, last checked 07/16/2001.

[19] “SourceForge.” `http://www.sourceforge.net/`, last checked 07/16/2001.

[20] “Apache Project.” `http://httpd.apache.org`, last checked 07/16/2001.

[21] “Qmail: A Replacement for Sendmail.” `http://www.qmail.org/top.html`, last checked 07/16/2001.

[22] “The MySQL AB.” `http://www.mysql.com`, last checked 07/16/2001.

[23] “Perl.” `http://www.perl.com`, last checked 07/16/2001.

[24] “Tcl Developer Site.” `http://www.scriptics.com/`, last checked 07/16/2001.

[25] “GNU Emacs.” `http://www.gnu.org/software/emacs/emacs.html`, last checked 07/16/2001.

[26] “K Desktop Environment Home.” `http://www.kde.org`, last checked 07/16/2001.

[27] K. Coar, “The Apache Web Server Documentation Project.” `http://apachetoday.com/news_story.php3?ltsn=2000-09-27-001-01-OP-CY-LF`, last checked 07/16/2001.

[28] “Mozilla Ongoing Localization Projects.” `http://www.mozilla.org/projects/intl/index.html`, last checked 07/16/2001.

[29] “KDE National Pages.” `http://www.kde.org/international/index.html`, last checked 07/16/2001.

[30] “The KDE Translation Teams.” `http://i18n.kde.org/teams/index.shtml`, last checked 07/16/2001.

[31] “Basis Technology Inc.” `http://www.basistech.com`, last checked 07/16/2001.

[32] L. Wall, T. Christiansen, and J. Orwant, *Programming Perl*. Sebastopol, CA: O’Reilly and Associates, Inc., 2000.

[33] “Perl Wrappers for ICU Project.” `http://picu.sourceforge.net/`, last checked 07/16/2001.

[34] “International Components for Unicode.” `http://oss.software.ibm.com/icu/index.html`, last checked 07/16/2001.

[35] B. B. Welch, *Practical Programming in Tcl and Tk.* Upper Saddle River, NJ: Prentice Hall, 2000.

[36] “The Mule Project.” `http://www.m17n.org/mule/`, last checked 07/16/2001.

[37] K. Handa, M. Nishikimi, S. Tomura, and N. Takahashi, “Unified and Extensible Mechanism for Multilingual Text Processing,” in *Workshop on Future Issues for Multilingual Text Processing, the 4th Pacific Rim Intl. Conf. on Artificial Intelligence*, (Tsukuba, Japan), 1996. `http://www.m17n.org/mule/pricai96/index.html`, last checked 07/16/2001.

[38] “The Sendmail Inc.” `http://www.sendmail.com`, last checked 07/16/2001.

[39] “The SoftAgency Co., Ltd.” `http://www.softagency.co.jp/index.en.html`, last checked 07/16/2001.

[40] “Commercial Support for qmail.” `http://web.qmail.org/top.html#paidsup`, last checked 07/16/2001.

[41] “Perl Established User Groups.” `http://www.pm.org/groups/`, last checked 07/16/2001.

[42] “Perl Workshop in Germany.” `http://www.perlworkshop.de/`, last checked 07/16/2001.

[43] J. Lerner and J. Tirole, “The Simple Economics of Open Source,” 2000. National Bureau of Economic Research Working Paper 7600, `http://dsl.nber.org/papers/W7600.pdf`, last checked 07/16/2001.

[44] “The Multilingualization Organization.” `http://www.m17n.org/`, last checked 07/16/2001.