Presentation Notes
Paper
Bio

P R E S E N T A T I O N

T4

Thursday, October 29, 1998
10:30AM

INTERNATIONALIZATION AND LOCALIZATION ISSUES IN TESTING

Jeffrey Jewell
Primavera Systems, Inc.

International Conference On
Software Testing, Analysis & Review
October 26-30, 1998 • *San Diego, CA*

Internationalization and Localization Issues in Software Testing

STAR'98*WEST*

Presented by Jeffrey H. Jewell

Primavera Systems, Inc.

Internationalization

- Preparing a product for the world
- Making the product localizable
- Separate locale dependent aspects of product from code base
- Involves both design and implementation

Why Internationalize?

- Global computer market
- Cost-effective localization
- Avoid retrofitting code
- Find and fix bugs earlier

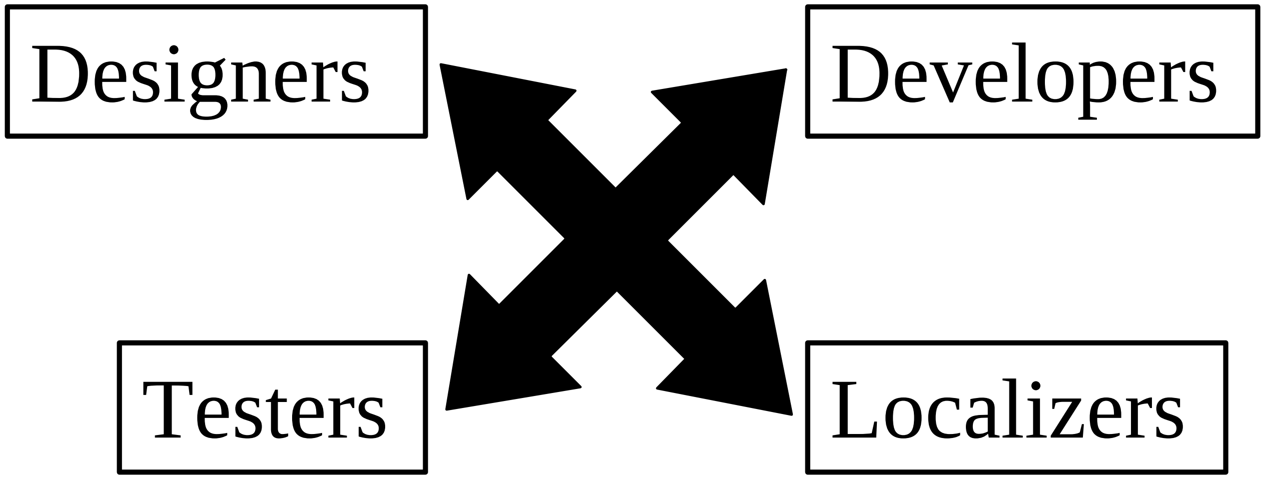
Localization

- Adapting a product for a specific locale
- Includes adaptation of:
  - Language
  - Culture
  - Customs
  - Standards

Areas of Localization

- GUI
- Help
- Setup
- Documentation
- Packaging

Who’s Responsible?



All must keep international objective in mind throughout the entire process!

Testing Internationalization

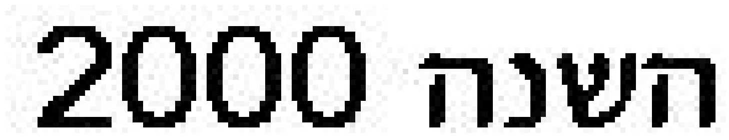
The Graphical User Interface

- Text & Controls
  - String lengths need room to grow
  - Number of items (controls in dialogs, menus)
  - All strings must be stored in separate files
  - Avoid combining strings

Testing Internationalization

More GUI

- Justification & Directionality
  - Text displayed and entered
    - left-to-right
    - right-to-left
    - top-to-bottom
  - Bidirectionality - numbers and foreign words
  - Controls usually oriented for one mindset



Testing Internationalization

More GUI

- Images
  - Beware of text on splash screens, toolbar buttons, and icons
- Sorting
  - Should be based on OS settings
  - Test in main data, dialogs, list boxes, etc.
  - Conventions with case, diacritics

Testing Internationalization

More GUI

- Printing
  - Test with foreign printers
  - International paper sizes
  - Units of measure
  - Fonts
  - Directionality
- If possible test with pseudo-translation

Testing Internationalization

Functionality

- Text Entry and Character Sets
  - ASCII
  - Upper ASCII (8-bit)
  - Double- and Multi-byte
  - Unicode
  - Use combinations of sets

Testing Internationalization

Formats and Standards

- Dates
- Times
- Numbers
- Currencies
- Separators
- Units of measure
- Calendars

Testing Internationalization

More Formats and Standards

Locale	**English (United States)**	**French (Standard)**	**Icelandic**
Currency	$123,456,789.00	123 456 789,00 F	123.456.789,00 kr.
Long Date	Thursday, August 20, 1998	jeudi 20 août 1998	20 ágúst 1998
Short Date	8/20/98	20/08/98	20.8.1998
Time	2:16 PM	14:16	14:16
List Separator	,	;	;

Testing Internationalization

Compatibility

- Operating Systems
  - Windows 95 has 3 different code bases
- Hardware
- OS Settings
  - Screen resolutions
  - Regional settings
  - Keyboard layouts

Testing Internationalization

More Compatibility

- Other software
  - IME support
  - Different language versions of products your product interacts with
  - Utilities (spelling, hyphenation, etc.)
- File operations
  - Names and paths in various character sets

Test Automation

- Follow same guidelines as developers
- Automate with localization in mind
- Refer to controls by other than label
- Separate strings from code
- Include international info in data sets

Testing Localized Software

Who?

- Bugs split between localizers & developers
- Get help from the locals
  - Use Beta testers, contractors from local area
  - Provide simple test plans and automation
- Leave translation to the experts
  - Ensure consistency
  - Ask for reverse translation

Testing Localized Software

How?

- Reuse test plans and automation
  - Get automation localized as well
- Compare versions side-by-side
- Use local default OS settings
- Swap files with English

Testing Localized Software

What?

- Perform full regression test cycle
- GUI
  - String lengths and placement
  - Menus
  - All appropriate strings are translated
- Compatibility
- Help

Conclusion

- Internationalization is a state of mind
- Do it the right way from the start
  - Waiting until localization costs more
- Other Sources:
  - Developing International Software for Windows 95 and Windows NT, Nadine Kano
  - “Multilingual Computing & Technology”

Internationalization and Localization Issues in Software Testing

Jeffrey H. Jewell
Lead Software Test Engineer
Primavera Systems, Inc.

The advances in computer technology over the last decade have brought the prices and appeal of computers within reach of a large portion of the world. As computers become more popular throughout the world more and more independent software vendors are hoping to capture some of that global market by making their software products available to consumers in foreign countries. This increased interest in international software sales means that software development teams are more involved in preparing products for use by people all over the world. Never has there been a greater need, then, for software testers to ensure the quality of software being prepared in many different languages, for many different locations.

The demands of a rapid application development environment and the modular focus of object oriented programming do not make it feasible or desirable for companies to develop completely separate products for different markets. Software developers want to get the most out the code they write, and minimize the amount of effort required to customize their programs to the demands of a specific location. This is where the processes of internationalization and localization take over, and where quality assurance personnel can make a significant contribution to that effort.

Internationalization and Localization

Software **Internationalization** is the process of preparing a product to be made available to the rest of the world. This process, also known as *globalization*, separates the pieces of a product that will need to be changed for international markets from those that should not be changed, thus making the product suitable for localization.

Properly internationalizing a software application minimizes the impact of localization on the application's code base, thus ensuring higher quality in the end product. Software that is developed with these principles in mind will not have to be "retrofitted", or internationalized after the fact, when it comes time to make it available to foreign consumers. Testing for software internationalization will allow testers to find bugs earlier than waiting until after the product is localized. Taking these measures will make the task of localization much easier to do, and less expensive in both time and money.

Once an application has been internationalized it is ready to be adapted for specific locales, or **localized**. Software localization involves customizing many different aspects of an application. Text must be translated into the local language and dialect, which seems to be the most obvious part of localization. But cultural variations between locations must also be accounted for, as well as local customs and standards. Specific examples of these changes and how to test them will be discussed later.

The changes that must be made when a product is localized affect all areas of the development process. Localizers must make changes to the graphical user interface, setup of the application, the help system, the printed and electronic documentation, and even the packaging. All of these things must be tested to make sure that the same quality is maintained throughout all the versions of the product.

All members of the development team are responsible for ensuring that a software product is properly internationalized, and it is something that must be kept in consideration throughout the entire development process. Designers, programmers, and testers will all have to remember that the product will eventually be used in global markets where the language and culture are different. Coordination between these teams along with the localization team is essential in maintaining a reliable product and an efficient development cycle.

Testing Internationalization

As pointed out earlier, testing that an application is properly internationalized will save time and money in the long run. It's definitely worth the time and effort to do it right the first time. Because so many areas of the product are affected by localization it is imperative that internationalization issues are considered in each test plan. Paying attention to the following specific areas will help testers verify the localizability of a product. This list is not all inclusive, but it does touch on many of the things that can cause big problems if they aren't taken care of early.

Graphical User Interface

The changes made to the text strings and controls of a product when it is localized can wreak havoc on the GUI. Testers must check to make sure that text labels and other strings in the GUI are appropriately sized, and will have room to grow when the product is localized. The less controls will have to be resized and moved around the better the localization process will proceed. Because it usually takes more words to express the same concept in a foreign language than in English be sure that there is space for more text if needed, and that dialogs are not too tightly packed that they can't be modified without major reconstruction. Also many foreign words will be longer than their English counterparts, which can especially affect the length of menus. An application that has just enough room for all it's menus to fit on one line in English may have to resort to using abbreviations in order to fit the menus on in another language. One application I worked on had just such a problem when translated to German and the result was the Help menu had to be translated as simply a question mark to save space.

Another way to ease the impact of localization is the store all text strings in external resource files, away from the actual code that uses them. This is a painful process to undertake after the product is well into development, but is fairly simple if this guideline is followed from the start. With all the strings stored in separate files only the files that need to be customized will be adjusted, minimizing the chance that bugs will be introduced into the code during localization. This also makes it easier to rebuild versions of the product. Testers should be involved in code reviews and can help spot text strings that have been missed.

A common programming practice that testers and developers must look out for and avoid at almost all costs is combining separate variable strings to form phrases or sentences. For many this seems like a harmless thing to do, but when localized these phrases can offend users or perhaps make no sense at all. Consider, for instance, the simple practice of using the user's name to show possession of an item, such as "Jeff's Document." This is a very simple thing to program by simply combining two strings: <username> & "'s Document." But in most other languages you can't show possession by simply adding a suffix to the noun as you can in English. In Spanish you would have to say "El documento de" & <username>, requiring a change to the code base. And what would you do if the combined string had to agree in gender with the user, as it would in some languages. It's best to not combine strings at all in this way. When it is necessary to concatenate words to make some compound idea use formats that can avoid number and gender agreement, such as saying "Interval: Weeks" rather than "Weekly Interval".

Special testing effort will also be needed to determine if an application can handle some other fundamental ways that languages differ. One of these is the direction a language is written in and read. Most languages are written from left-to-right, but for those near and far eastern languages that aren't it can be a real problem trying to work in an application that doesn't handle their language's directionality. Some languages like Arabic and Hebrew are written right-to-left, and others like Chinese can be written from top-to-bottom. A further complication comes with the concept of bi-directionality, which means that some things are written in one direction and the rest are written in another. For instance, in Hebrew and Arabic the standard text is written from right-to-left but numbers (yes, even though we call them Arabic numbers) are written the same as in English, from left-to-right, as are terms that are quoted but not translated from other languages. Installing on one of these operating systems will tell you rather quickly how well your application holds up, and you will need to test that all different types of characters are written in the appropriate direction.

Likewise application's tend to be designed and have controls arranged with the native locale's directionality in mind, such as placing a Next button on the right and a Previous button on the left. Although some of these designs will not have to be changed because computer users in foreign countries can learn to adapt themselves testers should still keep in mind these orientation issues when considering internationalization.

Another fundamental way that languages differ is in the way that things are sorted in a language, and it is essential to test that an application can sort according to the proper order for a specific locale. Sorting can be done according to the alphabet of the language, according to the number of characters in a symbol (like Traditional Chinese), or other means. Sorting rules can also vary with respect to upper vs. lowercase letters, characters with diacritics (accent marks), and other rules. For example in Spanish two l's together are a single letter so the word "loco" would appear alphabetically before the word "llave". Usually the operating system will provide rules for sorting, so testers need to verify that the application follows those rules. Sorting is something that is universal to almost all programs, and occurs in main data windows, dialogs, list boxes, etc.

A few more aspects of testing the GUI for internationalization need to be mentioned. Images with text on them can be a big problem because translating a picture is not as easy as translating a simple text string. You should try to minimize the amount of text that appears in icons, toolbar buttons, splash screens, and other images that will need to be changed when the product is localized.

Finally, testers need to make sure that an application can print properly for many different locales. Programs should support foreign printers and their drivers, international paper sizes, units of measure, fonts, and directionality. Failing to test these things can mean disaster for applications when they are localized.

A great way to test many of these aspects of GUI internationalization is to use a *pseudo-translation*, which "is a method of extracting text strings of source characters, replacing them with strings of characters from the target language [that don't necessarily need to make sense], modifying the length of the strings and inserting them back into the code." (Uren, p. 50) Using a pseudo-translation can give you a trial run at localizing a product without the expense of actual translation. Doing this can help you to see that all strings are translating, controls are appropriately sized, sorting rules are being followed, and other conditions are being met to ensure that when actual localization takes place fewer problems will be encountered.

Character Sets

One of the most challenging aspects of developing software for international use is testing that speakers of other languages will be able to input information into your application in an intuitive and natural way. Testers should also assure that the program will be able to properly handle the great range of characters and character combinations that people will use. Because languages don't use the same written symbols a variety of means have been established for programs to store and display these symbols, primarily using different character sets, which testers should focus on.

The ASCII character set consists of 256 ordered symbols with a numerical index, with each character capable of being represented by one byte. This is a sufficient number of characters for English and most of the European languages. However many Near and Far Eastern languages require more symbols than can be contained in this one set of characters, with some, like Chinese and Japanese, using thousands of symbols. These languages employ sets of characters using one or more bytes to represent the symbols, which are multi-byte character sets (MBCS).

The problems encountered when working with a MBCS could fill volumes, so careful attention must be paid to testing using these characters. The biggest problems come from programs that assume that each character is a single byte. For instance when you hit backspace to clear a character many programs that aren't multi-byte enabled will delete only the second byte of the character, thus changing the symbol that is displayed rather than deleting the entire character. In addition the selection of words, line breaking, and simple text editing will all suffer if the proper measures aren't taken to enable an application to work with MBCS systems. (For more info on testing multi-byte systems see Rollins' article.)

To test that a product correctly handles the complex array of characters testers should include combinations of various character sets in their test data. Any place that text can be entered and displayed should be tested with upper ASCII characters (those in the upper half of the set that require the eighth bit to be set), double-byte characters in a pure double-byte systems, and single- and double-byte characters in multi-byte systems. The areas that need this kind of focus include control labels, input fields, and file names and paths. Including character set testing in every test plan will prevent many embarrassing and costly errors later on.

The Unicode character encoding system, created and maintained by the Unicode Consortium, is a 16-bit worldwide set of code points that can be used to represent thousands of characters in a uniform system. Windows NT and other operating systems are now using Unicode which simplifies the coding procedures, but this system doesn't alleviate all problems because applications still need to have fonts available to display the selected characters. (For more info on character encoding see Kano, Chapter 3.)

Formats and Standards

Some locales may not differ at all in the language that is used but only in the standards and formats employed. For instance Windows 95 supports 75 locales, with a dozen languages that are used in multiple locales (see Kano, p. 3 and 9):

- Arabic
- Chines
- Dutch
- English
- French
- German
- Italian
- Norwegian
- Portuguese
- Romanian
- Russian
- Spanish

The regional settings for these languages contain dozens of settings that change from one locale to another, with little or no change to the language used for the locale, and many of these settings should be used for formatting in internationalized applications, rather than hardcoded standards.

The formats that must be tested most thoroughly are dates, times, numbers, currencies, separators, units of measure, and calendars. These formats need to be tested to ensure that items are placed in the correct order, the correct separator symbol is placed in the correct location, and that values entered in these formats are properly recognized. The following table shows some of the standards for three locales (see Kano, p. 3, for additional examples):

Locale	**English (United States)**	**French (Standard)**	**Icelandic**
Currency	$123,456,789.00	123 456 789,00 F	123.456.789,00 kr.
Long Date	Thursday, August 20, 1998	jeudi 20 août 1998	20 ágúst 1998
Short Date	8/20/98	20/08/98	20.8.1998
Time	2:16 PM	14:16	14:16
List Separator	,	;	;

Failing to adequately test formats can make your application more difficult to use and less attractive to foreign customers.

Compatibility

When testing for proper internationalization there are a number of additional compatibility tests that must be executed. The most important of these is testing in various localized operating systems. Windows 95 has 4 distinct code bases for different locales that share common development issues: European, Middle Eastern, Far Eastern, and Thai.

> "There were no code differences between language editions of Windows 95 in each of these categories—only the language of the user interface changed. The Middle Eastern, Far Eastern, and Thai code bases are all supersets of the European code base." (Kano, p. 9)

You should test your application in at least one language version of each of these categories to verify that the issues for that particular group of languages are addressed.

In addition to the operating systems themselves you should also test the specific regional settings within those operating systems. You should try to very that the changes associated with those regional settings are properly handled within your application. For instance if you change the date format while your application is running the dates displayed in your application may need to change as well. Other OS settings you will want to test include using different screen resolutions and keyboard layouts.

There are other applications that you should test with for compatibility. Asian languages use Input Method Editors (IMEs) to control the input of the thousands of possible characters. The default IME that ships with MBCS operating systems are usually the best but there are many third party versions that enjoy widespread use.

Finally, many applications interact with other utility programs and OLE Automation servers. You should test with localized versions of these applications to ensure there are no unforeseen problems. For example and Visual Basic program that is written to automate the English version of Microsoft Excel will not necessarily work with other language versions because the Excel object and method names were translated as well. In one product I worked on we were required to provide a localized version of the script that used German object names so that even users of our English product could work with German versions of Excel.

Test Automation

Automated testing can be one of the most valuable tools you can have in testing localized software, but it must be internationalized in order to be useful. Testers should follow many of the same guidelines that programmers follow in order to ensure that their automation can work with localized versions of their products.

For example, text strings in automation should be separated from the code so that, if needed, they can be easily localized. This will save you the trouble of having to perform wide-scale search and replace tasks on your scripts. To even further limit the amount of work to make your automation more portable you should try to refer to controls in your automation using values that don't change when the software is localized. Instead of referring to a Cancel button by the label "Cancel" (which would have to be translated) or by the location of the button (which may change during localization), reference the button using the control ID or index position.

A final suggestion for internationalizing your automation is to include international data in the data sets you use in your automation. Having your scripts use random characters to enter into the program can help you find a lot of things could otherwise be missed. You should also include data in specific localized formats in your general data tests, as well as creating separate and specific localized data sets. Automating with the end goal of localization in mind can make the process of testing localized software much easier.

Testing Localized Software

Approaching the testing of a product in a foreign language you don't know can be a very daunting task, and yet it is a frequent occurrence in the software industry. Getting some help from people who know the language can be easier than it seems and can provide invaluable aid in assessing the quality of the product as a whole, as well as the quality of the localization effort.

Who?

Just as programmers should not be completely relied upon to test their own code so should localizers not be completely relied upon to verify their work. Hiring contract testers from the target locale is one way to test localized versions. Another good way is to get Beta testers from the target area, people who already know your product or at least will know how it is to be used. These local users can help you find problems in translation that you probably won't be able to recognize. If you provide these testers with basic test plans and automation they can run they will be able to be of much greater service to you.

Because you don't know the language you're going to have to leave the verification of translation to the experts. But you can check to make sure that terminology is consistent by verifying that the same word appears in the same places throughout the product. You'll probably pick up a few things in the target language as you go along. If you really want to verify that the translations are accurate have a reverse translation done by another native speaker of the target language (other than the original localizers). You don't have to do the whole product but a good sampling of text translated from the target language back to the source language will let you know if the right messages are getting across.

When you find problems in the localized version you're going to have to be more careful about choosing who to report the bug to. Defects will be divided between developers and localizers. If your application has been properly internationalized it should be possible to isolate the specific file where the problem lies by replacing pre-localized files with those of the localized versions. If it's in the executable you're probably safe sending the bug to one of your programmers, but if it's in a resource file you'll probably want to report it to the localizers.

How?

In going about testing localized versions of software your best resource will be side-by-side comparisons of the original and localized versions of the application. These will not only help you know where you are going and what you are doing in the localized version but you will be able verify that things appear as they should.

As mentioned earlier your test automation can also be very useful in the localized version, especially if you can get the automation localized at the same time that the application under test is localized. Otherwise you'll be left with the job of replacing things yourself.

Finally, be sure to test localized versions in the local operating system, using the local OS settings and defaults. You want to test the software the way that local users will use it.

What?

In your testing of localized software you should plan to perform a full regression test cycle of the software. If you did a good enough job during internationalization testing of the original version you probably won't find many problems in the code that need to be changed, but you need to be sure just the same. Go through the same tests for compatibility, printing, and so forth to ensure that the product you're sending out is in the shape you want it.

You'll want to pay particular attention to the GUI, since that's where most of the changes are made during localization. Verify that strings lengths are appropriate because it's very easy for words to get cut off, and if you don't know the language it's harder to notice. Pay close attention to menus and controls, and test for all the same things brought up earlier for testing the localizability of the GUI.

You will also want to be careful that images and icons used are understandable to local audiences. A popular anecdote from the early days of computers is the confusion that arose from the Apple OS trash can that resided on the main OS screen. It seems that in some cultures this image resembled the local mailboxes and users frequently were upset to find that files they thought they were mailing were really being deleted.

Conclusion

Most software testers are probably becoming more keenly aware of the demands that our global computer market is placing on software development companies. As they are also more aware of the requirements of proper internationalization and localization of software they will be able to more easily meet those demands. Internationalization is a state of mind that must always be maintained in order to assure that the quality of software being delivered to local consumers meets expectations and standards. If internationalization is done correctly from the start, and tested thoroughly, the process of localization will be much faster and less expensive.

Bibliography and Suggested Readings

Kano, Nadine. 1995. *Developing International Software for Windows 95 and Windows NT*. Microsoft Press.

Rollins, W. Michael. 1998. "Testing Localized Asian Software". MultiLingual Computing & Technology, #20, Volume 9, Issue 4. Page 36. MultiLingual Computing, Inc.

Uren, Emmanuel. 1995. "Quality Assurance and Localization".   MultiLingual Computing, #9, Volume 6, Issue 2. Page 48. MultiLingual Computing, Inc.

Jeffrey H. Jewell

Jeff Jewell is a Lead Software Test Engineer at Primavera Systems, Inc. Primavera is the leading developer of project management software, including Primavera Project Planner and SureTrak Project Manager. Jeff has been testing with Primavera for over 3 years and has helped test and manage testing of products in English, German, Russian, Hebrew, and Japanese. He has had a key role in developing and implementing training programs for Visual Test automation and Visual Basic programming, specifically with OLE Automation. In addition, Jeff has been responsible for the project management of several software development projects. He has more than 7 years' experience working in the computer industry. Jeff's bachelor's degree in linguistics from Brigham Young University and minor in language and computers give him a unique insight into internationalization and localization considerations in software development. His years of study of Spanish and Hebrew and experiences living in Mexico also enable Jeff to better understand the intricate blend of cultures and languages that affect the software development process in a global environment.