Testing with an Accent

by Paul Carvalho

I BOUGHT MICROSOFT'S WINDOWS XP Home as soon as it came out several years ago. At the time, I received a free promotional package that included some third-party applications: an antivirus program, a firewall application, and a game of some kind.

I installed and set up Windows without a hitch and then proceeded to install the free software that I got along with it. A funny thing happened after I installed the antivirus program. Nothing. That is, the install seemed to be successful; however, no amount of clicking icons or coaxing executables could make the application run.

Oh, what to do? I decided to hit the Web to see if there were any updates to the software I had. Turns out there was a small patch for anyone installing the program on an "International English version of Windows." The description said that if you installed the program on an operating system not set to English (US), for example English (UK), then the program wouldn't open.

Fascinating. Until that moment, I had never seen such a bug.

My new WinXP system just happened to be configured with a regional setting other than the default. After all, why have the option of customizing your Windows "experience" and not take it? Not wanting to change my setup, I downloaded the patch and *voilà*, the program opened. Now that I had it working, I proceeded to think about how it might have been tested and how I might avoid missing such a bug when I test.

This bug is an example of an internationalization bug. In today's global software market, it's hard for me to think that any application I help test will be confined only to my local area and time zone.

The key here is *not* to think of internationalization and localization testing as extra work. Think of it as putting a different spin on the testing you already have to do. I've found that testing with a more global perspective in mind exposes bugs that you might otherwise miss. These bugs generally result from one of two things:

1. Changes to the system environment

2. Changes to the data inputs and outputs

Changes to the OS Settings

In this category, I place any change to the computer's operating system that takes place *before* you install the Application Under Test (AUT). Customizations to the operating system are often *not* taken into account by the development team.

There are many little settings that you can change in your operating system configuration that will make it more fit for use than going with the default options. These include setting the time zone, character set, country or region locale, currency, date and time formats, and so on. They may seem unremarkable at first glance, but just try changing some settings and see what impact there is on your AUT.

How do you know which settings to change? Change 'em all!

I usually leave one test machine with the default "out of the box" configuration so that we have a computer against which to compare bugs. With the other test systems, I give them "themes" and modify the OS settings to match. Themes might include setting up a system for an East or West Coast North American city, a different country, or someplace on a different continent. Look for interesting values when making these kinds of system changes. For example, if your standard date format is mm/dd/yyyy, change it to dd/mm/yyyy. Pick settings that look similar to a de-

SUSAN LeVAN

fault value but perhaps with certain values transposed or replaced. When deciding which values to select, I usually look for a pattern in the available choices and then pick the one that seems to stand out or break the pattern.

What can you expect when you make changes like these? Assuming that the AUT is well behaved and doesn't immediately choke when you install it (such as with my AV app bug described earlier), there are many possible outcomes depending on the level of internationalization and localization support designed into the program.

■ Changes to date, time, number, and currency formats should be automatically detected and applied throughout the AUT without error. Database queries based on any of these should continue to work as expected.

■ In a Client-Server testing environment, changing the time zones for the various workstations and server can expose interesting date-related issues. I once encountered a set of bugs specifically occurring once a year (in the fall) because the clock automatically turned back an hour on the server for Daylight Savings Time. So don't forget to check the scheduled time for your database backups or server maintenance tasks!

If you find yourself in a situation where you have translated language strings, request an internal "business expert" of some sort who is fluent in the particular language to validate the translations.

■ If a country or language locale setting has been changed, any language resource strings your AUT has should kick in automatically. The menu options should appear in the correct language, dialog boxes and windows should automatically resize to compensate for the different text labels, etc. This requires simple GUI walkthrough tests that anyone can do. (See Sidebar.)

If you actually have internationalization requirements that say you support operating systems in a different native language, you should install the supported operating system(s) in your test environment. Don't assume that changing the regional settings on a native English OS will be the same as running the OS in a different native language. There may be variations in the different OS versions that could expose bugs in your AUT.

Changes to the Data Inputs and Outputs

You don't have to make any changes to your operating system setup to detect potential internationalization bugs. Just consider the names you use for your "test users." I'm sorry, but "John Smith" just isn't a good choice. ("Jöhn Smîth" is better.)

I always include European characters (from the extended Latin character set) in my test user names. Try these special and accented characters in the User ID or User Name values (umlauts have been particularly good at breaking Login screens!). And there's always my personal favorite: "Miles O'Brian." It may look innocent, but names with single and double quotes often break unchecked SQL calls in various places.

I've found that many translation shops make many errors when they translate technical applications. One of my favorite localization bugs was seeing the "Exit" menu option translated as "This is the End"—not quite the same meaning and somewhat depressing. If you find yourself in a situation where you have translated language strings, request an internal "business expert" of some sort (perhaps a sales/marketing person or a consultant) who is fluent in the particular language to validate the translations. It's always been worth it for me.

What about ideogrammatic languages such as Chinese or Japanese? As with other tests, you need to consider how the data will be entered; what kinds of characters will be allowed; how this data will be used; and what sort of output is expected.

I tested an application once that stored and retrieved all double-byte characters entered by the user. The tests I did here were not much different from what I do with my simple European character tests:

■ Input something the AUT should handle.

Le Testing for Language Resource Support

If your application supports alternate languages via a Language Resource string component, there is a simple trick that you can use so that *any* tester can successfully identify localization bugs. Make a copy of the default language (e.g., English) string set and then alter it in some identifiable way. Be creative!

For example, we once called this new set "Le English" and placed the word "Le" at the beginning of each string or sentence. Then we changed the application setup so that it would use the "Le English" resource strings instead of the default language. A simple GUI walkthrough quickly identified all the places where the strings were missing and the text labels did not resize correctly.

Using a pseudo-language that's still readable in your native language lets you quickly identify any bugs or omissions while still allowing you to read the UI and use the application for testing. By spending some time keeping the new pseudo-language resource string set updated, your localization testing should integrate seamlessly with your other planned testing activities.

- Check to see how it is stored in the database.

- Find all the places where the data can be retrieved and displayed (e.g., text fields, combo drop-down boxes, title bars, reports, etc.).

- Find all the places where you can output the data (e.g., printers, files, filenames, images, etc.).

- Find all the places where the data can interact with third-party components. (Export some data and try to use it with a different program. Are there any command-line applications? There are usually lots of bugs lurking in feature interactions.)

Before you start testing, you should have an idea of the scope of language support that the application is supposed to handle. If you do not have explicitly stated internationalization requirements, then at the very least use some extended Latin characters in your data inputs when testing. With a little time spent upfront carefully selecting your test inputs and setting up your test environment, internationalization and localization testing should fall nicely into the background of your regular testing activities.

Before you start testing, you should have an idea of the scope of language support that the application is supposed to handle.

You don't have to know or speak another language to be able to do this kind of testing. It might help if you do, but it's not a requirement—that's a myth that may explain why so many testers I've known tended to shy away from these kinds of tests. Done properly, these tests should be fun and can expose a lot of really interesting bugs. With a little bit of applied effort, you might learn something new about a different culture or language—or even make new friends in a different part of the world.

Have fun. Learn. Explore! {end}

Paul Carvalho has been a software tester for more than ten years. He specializes in black box software testing and quality assurance. Paul can be reached at paul@staqs.com.

Sticky Notes

For more on the following topics go to www.stickyminds.com/bettersoftware

- Links to online I18N Testing resources
- Where to get ideas for Internationalized user names
- Where to download a FREE book on Software Testing with a chapter on Internationalization and Localization Testing