Opening the Black Box, Part II

Demystifying the Costs of Localization and Translation

By

John White
www.ventajamarketing.com

- Why is localization so expensive?
- Why does it take so much time?
- What can I do to lower translation costs?

Introduction

"Why is localization so expensive and time-consuming?!"

As described in Part I, we hear this question a lot from our clients. Try as we may, however, we find that there's just no getting around our somewhat ironical answer: "Words." Even when we've explained internationalization, translation memory and the processes that localization companies apply, as we did in Part I, we're still left with Words.

Now, in Part II, we'll explain why.

Too Many Words

This dimension is not hard to understand, though it's usually hard for most clients to believe.

"How could there possibly be so many words in the product?" they ask. "And even if there are that many, why should it add up to so much time and money?"

In the same way that "feature-creep" – small, incremental changes and product requirements – can suddenly add up to delays and cost overruns, word-creep can plague software and technical publications, even when the developers and writers work with the best of intentions for making the product easier to use and understand. In the original English, of course, it looks as though these additional sentences and paragraphs cost nothing: The developers and writers do not charge extra for each additional word, and clever formatting can prevent additional words from boosting the page count in printed material. Furthermore, nobody ever got promoted for releasing a version of the product with the lowest wordcount in company history, because it seems such an irrelevant metric.

However, wordcount is the cornerstone metric used by localization vendors, it is the one which usually figures most prominently in the cost of taking a localized product to market, and it is the one that first induces sticker-shock.

Some Solutions

Too Many Words is a structural complaint. "We don't know exactly why this costs so much and takes so long, but it must involve words. It's hard to identify and remove unneeded words, so we'll just say that, in general, our product has Too Many Words."

Depending on how many times the company has been unpleasantly surprised by the time and cost involved in localization, it has a different approach to the problem of Too Many Words.

New to localization – These companies complain about the high cost, but can rarely do anything about it because it's too late to make changes to the current version of the product. They may decide to:

- swallow the cost and chalk it up to experience
- pass some of the cost on to an overseas partner
- localize selectively (i.e., the client software but not the server software, or the quick reference but not the manual)
- oblige their prospective customers to use the English product

These companies also lose sight of the fact that the localized software will generate new revenue from new customers in new regions, and that the investment in localization is a necessary one. The erroneous perception that all of those words were "cost-free" in the original English – and, therefore, shouldn't cost much to translate – is typical of companies new to localization.

Some experience in localization – Companies in this phase begin to look more closely at ways to reduce the impact of localization costs, because they realize that their international products really are profitable, and they want to make them more profitable. Localization evangelists and project managers play a role in this stage by:

- explaining to developers and writers that more words today mean higher costs tomorrow
- reviewing product requirements early in the development cycle with an eye to selective localization
- looking deeper into each component of the product for words that don't need to be translated (see below)

By now, upper management sees that there is ongoing time-and-money pain associated with localization, that the pleasure of revenue outweighs the pain so far, and that further reductions in pain will require some serious changes in the company, even if management is not yet certain what those changes are or how they should be introduced.

Localization veterans – Companies in this phase work actively on integrating international and domestic product development, because they realize that "English is just another language" (EJAL). The integration process includes:

- putting in place and observing internationalization and localization requirements
- soliciting the localization vendor's perspective on process improvement
- streamlining and automating handoff of materials to vendors
- handing off pre-release content for a head-start on localization
- granting translators access to the global content management system
- implementing controlled language in the original text to facilitate automated translation

In this phase, the international products are part of the development process instead of an annoying adjunct to it. Companies only attain this stage once they have decided to let the localization tail wag the dog at least for a little while, until the dog and the tail see that each benefits from the other.

These measures, applied at different phases in a company's localization experience, address the structural problem of having Too Many Words in a product. They work at the levels of the overall product and the overall organization to reduce the number of words to be translated and the time to market. They are very effective, and project managers often refer to them when answering the question, "How much time and money *did* that save us?"

The next few sections focus on answering the question, "How much time and money *will* that save us?"

Too Many Words Repeated (In the Same Component)

The structure of some Web pages, help text and even software strings lends itself to repetition, especially when the structure includes a template that is used repeatedly.

- A manual describes dozens of similar applications in a suite. Each chapter follows a template containing standard paragraphs, modified for the name of the application.

```
                    Chapter 1
The following table lists the interfaces and controls
used in the development of [Application 1], along with
the files you need to run the application on the
handset. Before exploring the underlying code that
makes [Application 1] work, let's take a look at the
application from the user's perspective; that is, how
it works on a handset.
...
                    Chapter 2
The following table lists the interfaces and controls
used in the development of [Application 2], along with
the files you need to run the application on the
handset. Before exploring the underlying code that
makes [Application 2] work, let's take a look at the
application from the user's perspective; that is, how
it works on a handset.
...
                    Chapter n
The following table lists the interfaces and controls
used in the development of [Application n], along with
the files you need to run the application on the
handset. Before exploring the underlying code that
makes [Application n] work, let's take a look at the
application from the user's perspective; that is, how
it works on a handset.
...
```

The author of these paragraphs defends them in the name of user information, parallel structure and the instructional value of a certain degree of redundancy, but does s/he really need to use so many words? There will be some leverage and per-word discounts after the first occurrence has been translated, but too much of this parallel structure and redundancy becomes costly, especially over multiple languages. *That which appears free in English, costs money in other languages.*

- A programming reference describes hundreds of interfaces, and each interface generates return values. Technical Publications has prescribed to Engineering a template for uniformly describing these return values; however, it is the engineers themselves, not Technical Publications, who write the descriptions, and so the template is "more honored in the breach than the observance." Here are examples of `SUCCESS` and `ENOMEMORY` results from several different interfaces, as described by several different engineers:

```
SUCCESS Successful. Operation completed.
SUCCESS Successful. Operation is completed.
SUCCESS, If successful
SUCCESS: Execution successful
SUCCESS: if task is successful
SUCCESS on success


ENOMEMORY - There was not enough memory available to do the
operation.
ENOMEMORY, Insufficient memory.
ENOMEMORY: System out of memory
ENOMEMORY, if the implementation cannot allocate memory
internally for the operation.
ENOMEMORY: Not enough memory
ENOMEMORY:
ENOMEMORY: Memory error
ENOMEMORY: if enough memory couldn't be allocated
ENOMEMORY : ran out of memory while processing
ENOMEMORY: if memory could not be allocated to hold the
resource
ENOMEMORY: if Insufficient memory
ENOMEMORY, if failed
ENOMEMORY: can't add due to lack of memory
```

Assuming that there is no significant difference in the way that each engineer describes the `SUCCESS` or `ENOMEMORY` result, this is a nearly perfect waste of translation money.

Some Solutions

Too Many Words Repeated in the Same Component is a function of uncontrolled writing. Ways to reduce near- and no-matches in favor of 100%-matches include enforcing writing standards and editing ruthlessly. Some localization veterans have developed **controlled language** in their English original text to standardize writing, to make documentation more predictable for users and to render it more translatable.

Consider, however, the amount of time and effort required to get the developers to adhere to the template, and for QA of the English documentation to ensure uniformity.

Would the benefit outweigh the cost? There is money to be saved here, but it will cost money both to find it in the short run and to keep saving it in the long run.

Too Many Words Repeated (Across Multiple Components)

Single-sourcing and repurposing text are examples of maximizing leverage from content across multiple components in the product. If, for instance, a Web content team takes a large section of help text from the product and repurposes it verbatim in a page on the organization's support site, it is a victory for single-sourcing. When the organization hands the site and documentation off for localization, the 100%-matches between the help text and the Web page will help keep translation costs under control. However, there are some hazards associated with this practice:

- Fat Wordcount – When Management looks into localizing the product and the Web site, the combined wordcount will come as an unpleasant surprise, and nobody will remember that large parts of the Web content have been pulled straight out of the product. True, there will be no incremental translation cost for identical text, but there will be formatting and engineering costs.

- Different Translators – If the Web team should use a different localization agency from the one selected by the product team, it will be more difficult for each agency to take advantage of the other's translation memory. If this process is not carefully managed, then the organization will not only pay twice to translate the same English text, but also end up with different translations of the same English text (because performed by different translators).

- Change Management – If the Web team makes changes to the help text before posting it on the site, the Web content will no longer match the help text exactly. The decrease in exact matches will result in higher translation costs.

Here are some common examples of words repeated across multiple components.

- Software strings – The company whose software product evolves from a fat client to a thin client will probably try to get as much leverage as possible from existing resources when creating new ones. However, differences in platform and functionality will eventually result in resource sets that are no longer 100% matched to each other: A Web client, for example, will require error messages and configuration dialog boxes that do not apply to a Win32 client, and vice versa. When the time comes to hand off both clients for localization, the company may be surprised at how this bit of single-sourcing has broken down, especially as time goes by.

- Documentation – This is the area of greatest opportunity for re-use because it is where most words lie.

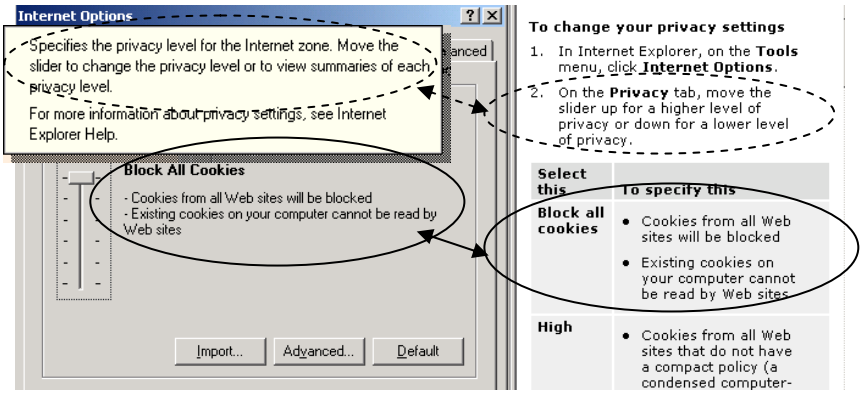

Figure 1

In Figure 1, the text in dashed ovals represents poor leverage (What's This help on the left and HTML Help on the right) which, with better single-sourcing, could have been 100%-matching text. The text in the solid ovals represents smart leverage from one component to another (software strings on the left and HTML Help on the right), in which 100%-matching text appears in both places. For translation wordcounts, the words in the solid ovals are repeated intelligently across multiple components, whereas those in dashed ovals are not.

- Web content – As pages and text are added to a Web site, the Web team faces the challenge of keeping track of all occurrences of similar text. If the team succeeds, then single-sourcing works well and 100%-matches keep translation costs down. However, when the text changes in one place and those changes are not propagated to all similar pages, it becomes an example of single-sourcing gone wrong.

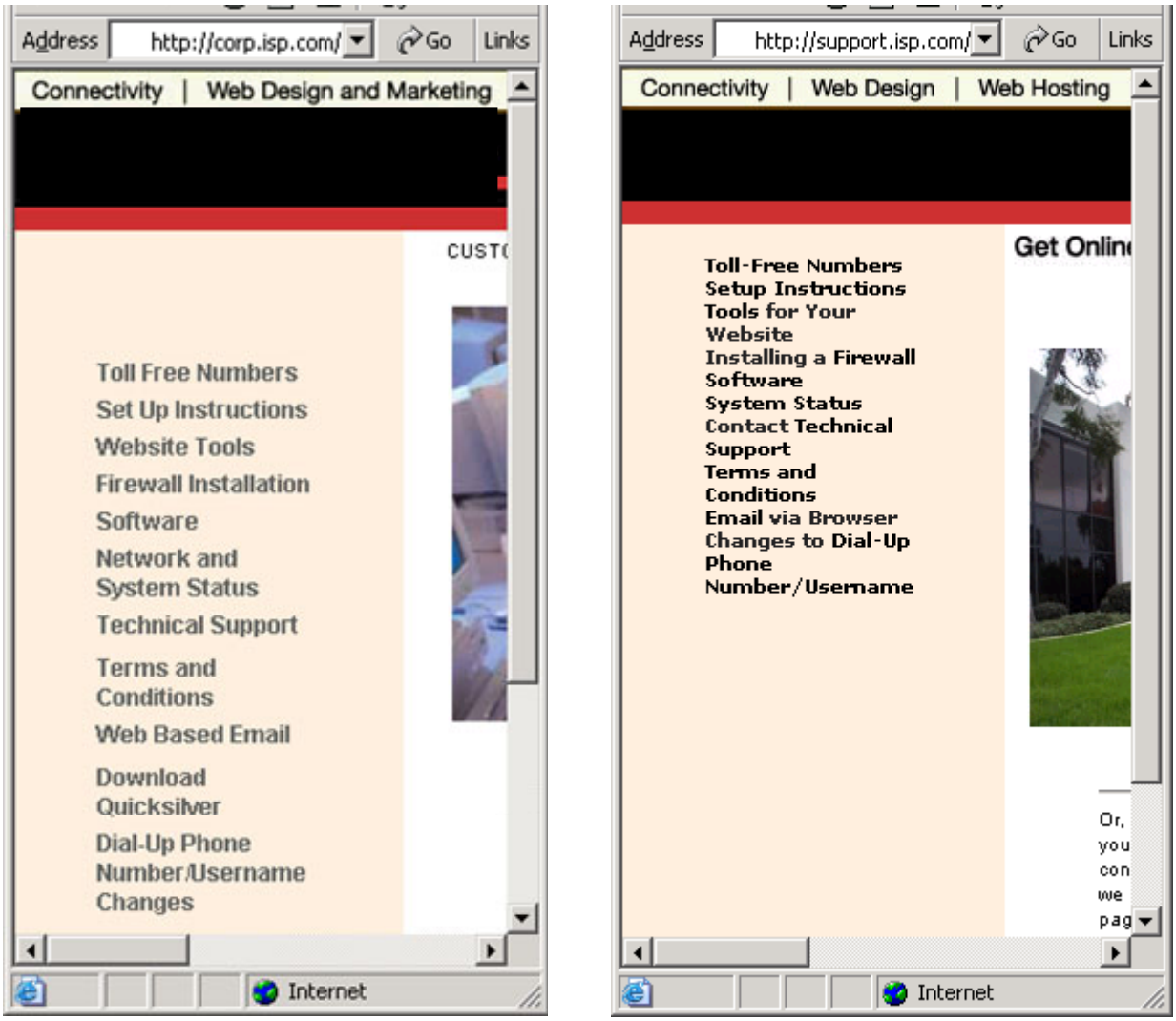

Figure 2

Figure 2 shows two instances of similar but no longer identical text in the left-nav of a Web page. Functionally, each set has the same set of links, but because single-sourcing has broken down, the labels have evolved differently. When the organization hands off its site for localization, there will be little leverage between these sets of strings.

Note that the URL on the left (corp.isp.com) is different from the one on the right (support.isp.com). Inconsistency across headers, footers and side-nav often occurs when separate Web teams – such as Internet, intranet and extranet groups – use the same elements without benefit of version control.

Some Solutions

Too Many Words Repeated Across Multiple Components is the result of good text moving too far too fast. When the time comes to localize it, it has moved a long way downstream and is in too many different places. A well defined, well implemented

strategy for single-sourcing is one solution to this problem, so that a given block of text is written with enough latitude to be useful in help, in software and on the Web without the need for modification.

Version control is also an important piece of the solution, because it tracks revisions to files and to text within files. Such a system can alert writers to changes and help them update repurposed text wherever in the organization it may reside.

The high-end solution is a **global content management system**, which stores both the English original and the translated text. If all translation is performed this far upstream then pushed out to all points at which the text appears, the problem of translating similar-but-not-identical text downstream goes away.

Too Many Words That Need Not Be Translated

Included in the wordcount of many technology products are words that need not be translated at all. Sometimes the text is obsolete; sometimes the text includes code or other literals which would only confuse users if translated; sometimes it should be localized, but not by technical translators.

- Software strings – As one version of software rolls to another and new functionality displaces old functionality, developers add and modify some software strings, but they rarely take time to discard obsolete text. Removing even obsolete text from a resource file can represent a change to the product which would need to be tested to ensure that removal did not introduce a bug, and most development efforts cannot afford such risk.

  Consider a resource file in version 5 of a product. It contains 10,000 translatable words, of which 2500 are obsolete yet still in the file. While it is true that the 2500 words have already been translated and will fall into the bucket of 100%-matches, it is nonetheless likely that the localization agency will charge for handling the words (engineering, QA) and that they will needlessly pass before the translator's and editor's eyes.

  Some developers address this by marking obsolete strings as "no translate" in a comment in the resource file, or in a readme file in the localization kit at handoff. Even if the localization agency finds the readme, and the translator obeys the instructions, the result will likely resemble a patchwork of translated and untranslated strings:

```
STRINGTABLE DISCARDABLE
BEGIN
    IDS_OBJECT_WIDTH        "幅"
    IDS_OBJECT_HEIGHT       "高さ"
    IDS_OBJECT_BPP          "ビット/ピクセル"
    IDS_STG_ERR_INVALID_IMAGE "The Object file %s does not exist."
    IDS_ERROR_BITMAPOPEN    "An error occurred while attempting to open
            the file %s."
```

```
    IDS_BITMAP_PARAM_SIZE_WARNING  "警告：選択した画像は、幅%dピクセル x
            高さ%dピクセルです。標準的な画像リソースは、幅%dピクセル x
            高さ%dピクセルより小さいものです。"
    IDS_BITMAP_DIALOG_NO_ID "You must enter an ID for this image
            resource."
    IDS_BITMAP_DIALOG_NO_NAME "You must enter a resource name for this
            image."
END
```

This introduces unnecessary costs. It is also bad software localization practice.

- Documentation – Many technical manuals and references contain long passages of text that need not be translated. In fact, translating a code excerpt such as

```
typedef enum
{
   AEET_NETWORK_LIST_AVAILABLE,
   AEET_NETWORK_LIST_USER_PREFERRED,
} AEETNetworkType;
```

would be a disservice to users, and any experienced technical translator would stay away from this text, even if not instructed to do so.

In a passage such as

```
#define AEE_START_OEM         // Launched externally by OEM software
#define AEE_START_SSAVER      // App launched as screen-saver

typedef struct
{
   int         error;         // Filled by app if there is an error...
   AEECLSID    clsApp;        // Class ID of the app started
   IDisplay *  pDisplay;      // Display class
   AEERect     rc;            // Rectangle for the App
} AEEAppStart;

#define EVTFLG_UNIQUE  0x0001   // Post only one of this type
#define EVTFLG_ASYNC   0x0002   // Event is asynchronous...
```

the text on the left of each line should not be translated, but the comments on the right of each line should be translated, so that users in other languages enjoy the same information as users in the English original.

While the human translators are smart enough to know what text to avoid, the software which counts words is not that smart, and, because the text is not in translation memory, the words will count at the no-match rate. The non-translatable text will then move to the translator, who will glance at it, realize there is no work to be done on it, and move on. Thus, the organization will end up paying for these words as if they had been translated.

- Web page keywords – Keywords live in meta-tags in the headers of Web pages. Depending on how the organization chooses to exploit search engines, the

keywords can be quite numerous. Unless specifically excluded, the keywords may end up in the wordcount. Since templates often include all keywords on all pages, this can boost translation wordcount of a Web site considerably.

Localizing keywords for localized Web sites can improve search engine results in global markets. A better approach, however, is to localize the keywords as part of a global marketing campaign. In-country marketing experts are better suited than technical translators to tuning keywords for local-market search engines.

Some Solutions

Too Many Words That Need Not Be Translated often occurs when the client assumes that wordcount will exclude and translators will ignore large passages of text that should not be affected by the translation process. The best way to trap such words is at the point of handoff from client to localization agency, but it is not always practical to review voluminous documentation or resource strings down to this level. Furthermore, some text, such as Web page keywords, is invisible without opening raw files.

While laborious, it's possible to locate the text with **visual inspection** and to comb the passages out by hand. If their occurrence is predictable, **Perl scripts** can identify them.

XML, too, holds promise as a solution. If authors create <noxlate> tags and label text accordingly, it will be much easier to find and exclude the text from wordcounts.

Summary

No matter the problem, the solution always comes back to deliberate efforts to decrease wordcount and to increase localization-awareness in the organization. The company's greater global reach calls for changes in process which, while costly, will benefit everybody. These efforts and changes may affect Engineering, Technical Publications, Web teams or all of these, and their goal is finding and eliminating words that needlessly drive translation costs upward.

Next Steps

You've learned something from this paper, haven't you? You'd like a strong process, plan and player for your localization project, wouldn't you?

To give your localization effort every chance of succeeding:

1. Help yourself to other resources on our Web site.
2. Become as conversant in localization terminology as you can.
3. Contact us for a **free assessment** of your project, before you paint yourself into any corners.

John White of venTAJA Marketing (johnw@ventajaNOSPAMmarketing.com) offers localization project management and international product management for technology companies, and has managed internationalization and localization projects since 1992. He sometimes wakes up in the middle of the night thinking about other places where Too Many Words lurk.