[*Translating and the computer 11*. Proceedings of a conference … 16-17 November 1989, ed. Catriona Picken (London: Aslib, 1990)]

Problems in software translation and how to avoid them

S. O'Sullivan

Translation Consultant, International Computers Limited, Language Translation Centre, Bracknell, Berkshire, UK

In considering software translation the following topics come naturally to mind: quality criteria, non-conformance to these criteria and the necessary solution.

What are the criteria for a good software translation? A good software translation from English into any other language should include no English words unless those words are in general use in the target language. The language used should be clear, the terminology consistent and the finished product be user-friendly.

However, only too often we find a number of non-conformances to this quality concept in the end product. Examples of this are:

— English words, in particular for commands and in menus
— truncated words and messages
— badly constructed messages, sometimes verging on the ridiculous
— ambiguities, in particular in the field of abbreviations.

Less problematic from a linguistic point of view generally, but irritating for the user, is the inability of software to cope with national characters. This can make words unwieldy and, in the case of commands, awkward to enter.

If experienced, conscientious translators have been used, most of these non-conformances will not be the fault of the translators, but will have been inherent in the English product. The main reasons for them are:

— text embedded in code
— field sizes that are inadequate and inflexible, i.e. that cannot be increased to accommodate the target language which almost invariably is longer

than English (e.g. German about 20 per cent., French often as much as 30 per cent.) and lack of information on field sizes.

— use of construct messages, i.e. messages that combine two or more elements, often not recognisable as such elements. For instance, you might find a message '%s not available from %t', or even worse, 'Cannot %t %s' with no information available as to what %s and %t stand for. Somewhere the terms which are replaced by those characters will be listed with no information that they are used in various contexts. This is fine for English where verbs and nouns often have the same form and where there is little or no gender and case dependency. In other languages such as German, French or Swedish this can often lead to chaos.

  Even when during a validation process badly constructed messages are found, it may be almost impossible to do anything about them due to grammatical problems. Examples of this are when a noun is used in constructions which entail it being once in the nominative, another time in the accusative and yet another time in the dative, or when an adjective would have to take the feminine gender in one instance and the masculine in another.

— Ambiguous abbreviations are usually due to restrictions on the number of characters permitted. Software designers and programmers are usually not instructed and not trained to take the needs of other languages into consideration and so they may decide that for the choice of background colour the first letter of the colour's name is to be entered. That is fine for English as long as you remember not to use blue and black. However, if you have the choice between green and yellow, in German this becomes a problem because green is *grün* and yellow is *gelb*, so you would have to enter G for both. Even three-letter abbreviations can cause a problem, such as when the respective words all start with 'sch' in German.

— Lack of information. Very often the translator is left in a void. He or she is given little information about the product, hands-on experience is not available, no contact person has been appointed. This is particularly true when working through agencies, or when designers and programmers are too busy to find all the answers.

  On the other hand, software design and development are only too often unaware of the requirements of the target language for accented or special characters; of the fact that the grammatical structure of the target language might be totally different from the structure of the English language, and that most languages require more space than English.

Although the problems listed are formidable, they could and should be overcome. The solution lies in cooperation, guidelines and management commitment.

Cooperation must exist between management, development and design and the translators. Ideally, this cooperation should start before the English product is being developed and should continue throughout development and translation. It is essential that everybody concerned is aware of the important role that the translator can play even at the development stage. This does not mean, however, that translation should start at the same time as the development of a product - far from it. The actual translation work should not start until the software is frozen or as good as frozen.

This cooperation should result in guidelines to be used for future projects as well. Included in those guidelines should be the following recommendations:

— Text should not be embedded in code, but should be maintained in a separate catalogue which makes translating, editing and later updates easier.
— Where product size permits, no construct messages should be used. Where such messages have to be used, the translators should be informed how the various elements are combined and where to find them.
— Field sizes should be adjustable and where this is impossible, software writers should be instructed to leave enough space for language expansion. For example they should not write an English message which takes up the whole space but leave about 30 per cent, of the field free. Any restrictions of field sizes should be documented.
— One letter abbreviations should be avoided at all cost. In general abbreviations mean trouble and should be used only when absolutely necessary.
— Last but not least, that there should be a constant information flow between all concerned. This exchange of information could be achieved through meetings, including progress meetings, as well as one-to-one conversations, reports and plans.

These guidelines should also be used when evaluating third party products prior to buying them in for further development. Such products could well be unsuitable for translation for the reasons already mentioned. CCI, now part of ICL, has taken these guidelines further and has developed tools with which all message strings are put into a database, combined with the appropriate language file where available. The translator calls this up in the form of a datafile with each string contained in a record. The record also contains any relevant information entered either by development or by previous translators. The usual search and mark facilities can be used, thus ensuring consistency and speed while maintaining flexibility.

Forms are also presented as datafiles which can be edited by word processing. This, combined with flexible field sizes and the possibility of simulating a software ‘build’, allows the translator to redesign forms where necessary.

The problem of substitution of nouns is tackled with gender and object tables which are prepared before the actual translation starts. The object table contains

the object names, e.g. file, folder, document, etc., both in English and the target language, in both singular and plural, as well as relevant gender information. The gender table contains the grammatical rules and the translations of articles. By using special function keys and index facilities the two are combined, so that the message 'The %t does not exist' with 'the' having the index 5 in the gender table, for instance, will read '%5 %t n'existe pas' in the translated string. The system will then substitute %5 accordingly.

Just as important is management commitment. Management has to be aware of the problems and requirements in the planning stages, in order to avoid setting impossible deadlines and to be able to commit the necessary resources. After all, software translation is not cheap. It requires highly experienced translators who are usually very busy indeed because there are very few translators willing to work long-term on-site and who therefore do not come cheap. However, in our experience only working on-site can lead to a really good software translation. A compromise might be achieved by providing a working system for the translators at home and giving them all possible help, in particular by appointing a contact who will deal with all questions and arrange any site visits necessary or will travel to the translators to give help and guidance.

Software translation, like any other translation, should never be rushed and adequate additional time should be allowed for the training of the translator, trial builds and subsequent changes. While a translation might look perfect on paper it might very well prove inadequate or wrong once seen in a working context.

A time-saving solution would be to provide tools with which the translators can simulate a build on their system and thus see the results of their translation more or less immediately and in context.

Software developers and programmers have to be convinced of management's commitment to the guidelines and to the resources required to implement them. Failing this, and as a result of other pressures, the developers will take shortcuts and create problems for the translators and eventually for themselves.

The old ICL motto 'We should be talking to each other' is certainly true in the minefield of software translation. Only by understanding and cooperation between the three main groups in the production of software, i.e. management, software design and development, and translators, will we achieve our aim, namely truly international, user-friendly products.

AUTHOR

Siegrun O'Sullivan, Translation Department, International Computer Centre, Lovelace Road, Bracknell RG12 4SN, UK.