Current Issues in Software Internationalisation

James M. Hogan Chris Ho-Stuart Binh Pham
Centre for Information Technology Innovation
Faculty of Information Technology
Queensland University of Technology
GPO Box 2434, Brisbane, QLD, 4001, AUSTRALIA
{j.hogan,c.hostuart,b.pham@qut.edu.au}

Abstract

The trend toward globalisation of products and services has brought a strong economic imperative to the development of general methods for the localisation of software to different cultural environments. While ad hoc, bolt-on localisation may satisfy immediate commercial objectives, its extension to multiple locales is not cost-effective and an integrated strategy is needed. In this more sustainable approach, known as software internationalisation, the requirements of disparate markets are addressed during analysis and system design, with the architecture developed so that localisation to a particular environment is straightforward, and involves minimal re-engineering.

Given the limited size of the Australasian market, detailed attention to the technical issues of internationalisation is of critical importance to the future of software development in the region, as is the availability of graduates adequately prepared for this environment. Thus motivated, this paper examines the state of play in a number of aspects of application level software internationalisation, with our focus the core research challenges of the next few years, and the consequences of these trends for the software engineering curriculum.

1. Introduction

Historically, commercial computer software has been developed for the English speaking community – dominated of course by the United States of America – with only a limited attempt to cater for other environments. However, globalisation of business and technology have heightened demand for software products *localised* to a particular language and cultural environment – demand which has been exacerbated by the spectacular growth of the world wide web. Minimally, such localisation must include translation of user interface strings to the target language, and adaptation of display formats to comply with local conventions for items such as date, time and currency.

Many of these issues are readily handled through standard programming language support, but others are surprisingly subtle, and even string translation has hidden complexity – such as that which arises through the absence of an identifiable equivalent term in the target language. When the process is extended to include cultural variations in the use of symbols and colour, and adaptation to local business processes such as the taxation regime, it is clear that successful localisation depends upon a careful and time-consuming blend of technical and native professional expertise.

Yet if the cost of localisation for a particular environment appears high, the expense becomes unsustainable if incurred for each new language and locale, and some amortization over multiple adaptations is essential if the process is to remain commercially viable. This more systematic approach, termed here *software internationalisation*, requires that localisation issues be addressed earlier in the software development lifecycle, with architectural and subsequent design provisions to minimise re-engineering.

While internationalisation has necessarily been addressed by major international vendors at both the operating system and key application level, much of this work remains commercially sensitive. To date, relatively little effort has been directed toward the establishment of *open*, industry-standard methodologies and documented practice – seemingly an essential pre-requisite if small and medium size enterprises (SMEs) are to compete in the global market. Moreover, such reports as are available suggest that even the 'advanced' organisations may suffer from poorly managed communication with, and utilisation of, professional translation services. In particular, there appears to have been almost no attempt made to integrate the internationalisation workflow within the tight development cycles of the modern software engineering environment.

It is perhaps self-evident that solutions to these problems must be centred around the fundamental design principles of good software engineering – modularity and re-usability – but this does not diminish the need for a clearer understanding of their application in the present context. Such principles will naturally inform key technological decisions – for example the division of localisation tasks between source code modification, link phase binding and run time resource loading – but these and other aspects of the process are equally driven by the pressures of cost-effective quality assurance. And given the dependence of successful localisation upon the approval of the native speaker, it is this latter factor which is in many respects the more influential, determining the prerequisite skill set and background of translation staff, and even the degree of tool support acceptable to the vendor.

In this paper, we shall limit our focus to the core issues of process and tool support for application development, with our objective the identification of the key challenges posed to the computer science community by software internationalisation. Such sentiments notwithstanding, we must immediately insert the caveat that our treatment is slanted toward those issues which we believe are amenable to a technical solution or simplification. Central among these are the problem of text translation (section 2), and the process issues of architecture and language support (section 3.1), and quality assurance (section 3.2). The fundamental problem of cultural adaptation is here treated somewhat superficially (section 4), largely as a result of the almost unbounded scope of the research problem. Nevertheless, significant progress is possible in this area through appropriate international standards, and this relationship is explored as part of (section 5), in which such agreements are assessed with respect to their effectiveness in delivering a common framework accessible to the SMEs which dominate the regional industry. This coverage sets the scene for discussion of perhaps the most important challenge to the profession – that of equipping software engineering graduates with the skills necessary to flourish in the internationalised environment (section 6). We conclude with some guarded speculation on the future of software internationalisation in the region.

2. Translation and Tools

Issues of text translation lie at the heart of software internationalisation and localisation, and the health of the discipline may in many respects be measured by progress in this area. Developments in translation and tools are necessarily linked to developments in programming language and operating systems support, and in the growth of internationalisation support in these areas serves to limit the core research problems remaining in this area.

While tool support is an integral part of modern translation, it must be emphasised from the outset that the problem of machine translation remains extremely difficult, and the limitations of existing systems may be readily demonstrated by performing a return translation[1] using the publicly available Google translation beta (Goo n.d.). Moreover, the use of machine translation in software internationalisation is likely to prove problematic even when the dreams of the language technology researchers have been realised, due to the unusually high frequency in software translation work of novel word forms and combinations for which no ready equivalent may exist in the target language. Indeed, this problem may manifest itself in the selection of appropriately intuitive key or user interface button labels, and the value of the experienced translator over the novice may lie in the quality of this kind of selection (Dohler 1997).

Thus, since it is apparent that localisation to a particular region cannot proceed at present without substantial involvement from human translators, the goal of the efficient software vendor remains one of optimising the use of such services wherever possible.

In an earlier phase of the globalisation of the software market, text translation typically proceeded through an ad hoc attempt to localise existing software designed solely for the English speaking – and especially American – world. Such software is by definition poorly architected from an internationalisation perspective, involving substantial redesign of the product source code in order to isolate language and locale sensitive material. Moreover, localisation was initially fragmented, driven by the commercial need for penetration in a particular region, and in consequence requiring a limited range of language expertise. Subsequent localisation was often characterised by duplication of effort, and suboptimal usage of translation services. Tool support for such a process was similarly ad hoc, perhaps focused more upon re-engineering the product architecture than upon support of the process of translation.

While there are numerous small vendors who remain at what might be described by analogy with the Capability Maturity Model (The CMMI Product Team 2002) as Level 1 internationalisation software firms, the practice of software internationalisation has evolved substantially - at least to the point that major software vendors have addressed the most fundamental architectural issues, and modern programming languages and operating systems now provide direct support for the process.

Prominent software vendors have developed the following strategies for minimising the impact of internationalisation issues in their products:

- The extraction or "externalisation" of user interface strings into resource bundles and message tables – thus limiting the task of the translation service to that of direct translation of a database of known strings;
- Subsequent careful control and identification of the original strings, their translated target language equivalents and the *context* associated with each translation performed;
- Almost universal outsourcing of translation tasks to specialist software industry translation services, often based in key internationalisation centres such as Dublin and Beijing; and
- "Dumbing down" or simplification of string content and associated context in preparation for submission to the translation services or machine translators.

A number of these strategies are considered in more detail below.

Externalisation of UI Strings: Extraction of user interface strings and other language sensitive material to resource bundles is now standard practice in the industry, with only those vendors content with a single, monolingual market failing this initial test. Yet, again the apparently straightforward translation task is complicated by the embedding of markup codes (such as those used in RTF-based Windows help files) and hyperlinks within the supplied files. As noted above, the experienced translator has an important role to play in coining appropriate novel word combinations and even neologisms in the target language corresponding to the intention of the original. This practice is often hampered by the limited provision of system specifications to the translation service, and by the pressures of time to market – in which a "mediocre translation . . . delivered on time is much more valuable than a perfect one . . . which is three days late" (Dohler 1997).

[1] A translation of a text fragment into a foreign language and back to the native tongue. Note in particular that such limitations may become apparent even with short sentences.

Maintenance of a String Database: The benefits of this strategy are self evident, as translation costs are geared to the number of words in the original document - with prices ranging between US$0.30 and US$1.00 per word (Lerner 1999)[2]. However, optimism must be tempered by the realisation that the usefulness of the individual string is governed by the independence of its translation from surrounding context.

This area represents an important opportunity for research, through the investigation and development of intelligent and multi-lingually aware systems for the management of source string databases. Of principal concern must be the robust and *context and grammatically sensitive* identification of similarities between novel source language strings, and those for which a translation is registered in the database. Appropriate technology of this kind has the potential to realise enormous savings in translation service costs, and in the reduction of time to market for new systems.

Preparation of text for translation: This preprocessing step is regarded as particularly important in the context of machine translation, as the reduction of ambiguities greatly simplifies the task and enhances the reliability of the underlying grammatical model. The approach may also assist translation service staff, removing potential ambiguity due to issues such as unfamiliar colloquialisms, and reducing the word count.

One major machine translation vendor summarises these issues nicely in their recommendation to check language and sentence structure prior to translation:

> "Try to eliminate or alter sentences and phrases that have multiple interpretations. Remove unneeded words from a sentence or split a long sentence into two shorter ones" (Lan n.d.).

The language technology community have investigated similar issues through the development of a computationally sound simplified English, such as the "Controlled English" of Mollá and Schwitter (Mollá & Schwitter 2001). While acceptable Controlled English may be too restrictive for all applications – indeed the construction of the equivalent CE form requires translation of plain text into logical forms – the approach is nevertheless a important pointer to the future of assisted and ultimately automated translation.

While this analysis is necessarily speculative, it is likely that an important pre-requisite for the widespread adoption of such an approach will be the availability of target language specific text generation frameworks, which will act as a complementary service to any text simplification engine on the source side – or perhaps as a client drawing upon a repository of a more formal semantic specification such as that discussed above. This kind of technology has been adopted in a number of industry applications, with one of the better known systems being the CSIRO's Isolde project (Paris 2002).

Discussion: In summary, the problem of text translation remains at the core of internationalisation efforts, and there appears little prospect of a fully automated solution to the problem in the immediate future. While assisted translation applications are already available, the most important problem at this point lies in string maintenance and the avoidance of unnecessary and costly translation. To date, this effort appears to have been limited to careful indexing strategies and to our knowledge there are no applications in existence which perform intelligent trawling of an existing database for useful substrings – existing translations which may be exploited as part of the task even if there are differences in the original context.

Evidently, such a project shares a number of the difficulties encountered by the machine translation community, but these are ameliorated to some degree by the reduction in scope, and by the ready acceptance of a failure condition should an appropriate confidence prove unattainable for the given query. Moreover, the costs of translation suggest that even modest, incremental reductions in the translation task will be welcomed, and that integration of such a system with appropriate text generation facilities will bring enormous interest from the industry.

Yet such systems are of little use if the advantages they provide are overwhelmed by the cost of additional translations resulting from poor software engineering practice. These issues are considered in the following sections.

3. Software Engineering Practice

3.1 Programming Languages and Software Architecture

While our focus in this and the subsequent section is upon issues of software engineering process, the selection of an appropriate software architecture cannot entirely be divorced from the class of implementation language, and it is perhaps useful in any case to consider briefly the level of internationalisation support provided by industry standard languages. As will become clear, such support has advanced to the point that many of the key research problems in this area have been solved, with the convenient input of pictographic characters the one glaring exception[3].

The development of integrated programming language support for internationalisation may be traced to the release of the Java programming language in the mid 1990s, and we shall consider its facilities as representative, referring to competing products only as necessary. Java was the first mainstream offering to incorporate 16-bit Unicode standard (The Unicode Consortium 2000) characters as part of the language. Moreover, the associated class libraries provided extensive support for a substantial number of ISO standards for locales, incorporating date, number format, currency and language specifications.

While usage of these classes appears somewhat cumbersome – with the necessity of constructing both `locale` and `formatter` objects prior to output of a currency value, for example – the additional line or two of code seems a small price to pay for the inherent utility of the approach. In any case, the tutorial material available with the SDK and the considerable number of professional references on the market each provide appropriate test programmes, with

[2] Indeed, the authors are aware that one leading vendor – in spite of careful attention to limiting the need for new translation – paid almost US$10 million to translation services during their most recent upgrade, involving more than two dozen target languages.

[3] As noted earlier, our focus here is upon applications development and this issue is best regarded as an operating systems issue – and indeed the problem is receiving a good deal of attention from both Microsoft and the linux community.

boilerplate code which may be readily integrated into more elaborate applications.

Internationalisation support in Java is not, however, limited to these more obvious considerations (Horstmann, & Cornell 1999) The `locale` object may be used as a passport to localised behaviour in the following domains:

- Sorting and Ordering at various levels through a `collation` object;
- Text boundaries, through a `BreakIterator` object; and
- Management of resource bundles, through a `ResourceBundle` object.

The advent of similar support within the latest release languages from vendors such as MicroSoft (through C# and the availability of associated class libraries to managed code C++) serves only to emphasise the earlier point that most of the important challenges in this area have now been addressed and the fruits of this research are now widely available. While it would appear that there is some scope for further extensions to locale support, perhaps even at the level of tables of lexical equivalence, such resources are extremely unlikely to be integrated within the fabric of the language in the same way as the Unicode characters are in Java. Thus, even if such facilities are supplied by the language vendor, they cannot in practice be distinguished from third party suppliers offering support libraries for a multitude of purposes. In this sense, we are travelling beyond the scope of the programming language per se, and are leading naturally into the domain of the software architect. In many respects our path travels by way of the software component, and we shall return to this theme in due course.

Much of the available literature would suggest that progress in internationalisation sensitive software architectures has been limited – and indeed this impression is in part a consequence of the paucity of articles. In this discussion we will analyse the two extremes of the debate, offering some limited resolution of the conflict.

On the one hand, internationalised software could be based on general purpose processing engines, which can then access and use tables of information specific to a particular location to give appropriate behaviour in that context. This has the advantage that the software can quickly be adapted to new locations, and gives the possibility of a single binary release which works everywhere, since it is only the tables which need to be modified, not the processing engine itself. This style of architecture is proposed by Kokkotos and Spyropoulos (Kokkotos & Spyropoulos 1997).

The central limitation of this approach lies in its capacity for scope creep – resulting in process engines of byzantine complexity. The nature of internationalisation ensures substantial risk from the tendency to anticipate problems which may not have been demonstrated to occur, with a consequent risk of grossly inappropriate levels of functionality.

At the other extreme lies the pure minimalist approach, in which the application is at the outset designed as far as possible to exclude consideration of locale dependencies, with the initial focus upon core functionality. In principle, localisation support is then introduced as a goal of subsequent refinements of the product. Such an approach would appear well suited to the light weight, highly iterative processes now gaining wide industry acceptance for software development; the “agile” process movement.

A disadvantage here is that even a light weight process can become slow and expensive if it requires repeated rework for each new circumstance. The usual solution, in an agile framework, is to *refactor* parts of a product only at those points when it becomes clear what portion of a system is being repeated or revised on multiple occasions, and then refactor that portion to be more generally applicable. For software internationalisation support, it appears unlikely that such a bottom-up solution will prove optimal, and indeed internationalisation may provide a useful example for attacks upon methodologies such as XP, which is often criticised for its dependence upon ‘emergent’ architectural design.

It seems plausible that the optimal approach lies in the middle ground between such extremes, but it is difficult to discuss the general case without degenerating into banality. Plainly, in designing for a global market a developer needs to maintain a particularly clean separation of concerns between the human-computer interface, and the universally applicable data processes that may exist; and this requires due care and from the outset.

On the other hand, there remains a clear benefit to simplifying the interface of globalised applications as much as possible, thus providing some natural bound upon the complexity of adaptation.

Ultimately, it is our belief that the best solution to the architectural problem lies not in the specification of the internationalisation structure per se, but rather in the specification of a standard interface for internationalisation components, so that developers may phrase their localisation requirements in terms of appropriate messages through the necessary API. In one sense, we are proposing that the *industry* adopt coverall architectures in the style of Kokkotos and Spyropoulos, with developers able to select only those components necessary for their application. Moreover, such an approach offers the promise of a healthy third party software component market, particularly in important business domains.

Such applications will in many respects provide a fundamental test of the viability of the component market. Different locations may involve markedly different legislation, regulations and taxation regimes with a substantial impact upon the processing that must be performed. For software to be adapted easily to a new location, it should be easy to alter the business rules by which it operates, and this in turn suggests that even some underlying processing rules should not be hard coded into the application, but be included through some external mechanism. Ideally of course, the developers should be able to purchase off-the-shelf components which embody the regime in which they wish to operate, but even with the best of intentions this will never be the complete solution for a complex application. Some mixture of off-the-shelf and in-house internationalisation components would appear to be unavoidable.

Yet the problem of an appropriate software architecture is also linked to the important question of integration of internationalisation workflows with modern software engineering processes – a problem which we shall consider from the perspective of quality assurance in the next section.

3.2 Quality Assurance

The obvious principle that one cannot test quality into software has a clear corollary for internationalisation: one cannot translate internationalisation into software. Internationalisation issues need to be considered from conception through to packaging and delivery, and there are many distinct aspects to be addressed.

The development of maturity models for development processes has proven highly successful in managing the quality of software products. The Capability Maturity Model for software(Paulk, Curtis, Chrissis & Weber 1993), developed by Watts Humphrey and the Software Engineering Institute, has been influential in this regard, and related models have since been introduced to cover a number of additional aspects of software production.

Given the inherent complexity of internationalisation, and its potential impact upon the development process – from requirements analysis right through to the tasks of support and maintenance of the delivered product – we believe that this is a problem which is well suited to a comprehensive maturity model.

Martin Pol has developed a model for Test Process Improvement(Koomen, Pol, Broeders & Voorthuyzen n.d.), in which there are twenty key areas identified, and a test maturity matrix where levels for each of the twenty areas are identified and ranked. Such a model can be used to give a comprehensive picture of the maturity of testing within an organisation, so as to identify areas of weakness and opportunities for improvement.

There are good reasons for believing that a similar style of approach would work well for internationalisation. There are some obvious key areas which could be identified: string management, translation, acceptance testing in target environments, internationalised support, change management across target environments, matching of initial requirements to different international contexts, and so on.

A study of companies which have been successful in developing a product for diverse international markets is important to guide the identification of key areas and maturity levels for such a model, and conversely, the goal of producing a model in this form gives a useful framework for such a study.

An inevitable difficulty with this approach lies in the protection of the commercial interests of the participant company; those companies which demonstrate high maturity in this area may consider this to give them a critical strategic advantage over competitors, and so they may be cautious of sharing. On the other hand, one might reasonably argue that an improvement in the profile and reputation of the regional industry will be of benefit to *all* of its developers, and for some companies there may be a happy coincidence of altruism and self-interest.

As hinted at the start of this section, One guiding principle of any such model is the realisation that quality control for internationalisation cannot be addressed solely by testing a finished product for a foreign market. Nevertheless, such testing remains important, of course! When an internationalised product has been altered in any way as part of adaptation to a particular locale, full regression testing should always be performed on the final product. Moreover, the background and skill set of the testing staff must be carefully chosen, and these standards adhered to stringently. In this respect, there can be no substitute for third party evaluation by experts who are native to the target culture and language. Formal certification of third party evaluation or testing services would help, and there are some relevant standards that apply, such as ISO 2384:1977(International Standards Organisation 1997), which describes how translations should be presented.

One potential advantage for Australian software developers is that our Universities attract a steady stream of competent young people from overseas who are native speakers of languages which are likely to be targetted by software developers. There are obvious advantages to having software engineers in a company who are native to these countries. This is not a substitute for a carefully controlled process, but it may help make quality control more efficient and cost effective.

While all of these matters are properly considered as quality control for localisation, internationalisation takes the process one step further, and aims to be easily and naturally localised for different cultural and language contexts – with perhaps a greater likelihood that changes may be made at different stages of the life cycle and by different people. Particular care needs to be given to consistency in the case of distributed applications working at once in several diverse locales. This is one aspect of internationalisation which has not received a great deal of attention.

In the preceding paragraphs we have examined a number of specific aspects of internationalisation and localisation through the lens of traditional notions of quality control. While the adoption – after due adaptation – of best practice for quality control and assurance in Software Engineering is of critical importance, it is perhaps in this domain that issues of process communication and documentation become most important.

Quality assurance for internationalisation requires particular attention to the presentation and maintenance of data and documentation, the objective a clear separation of concerns between the general problem or task addressed by the software, and issues that relate to the choice of locale.

It is a well understood axiom of effective quality management that it is not enough to address the matter from a technical perspective; it is also important to focus on the attitudes and interests of people in the organisation. Personal issues may become particularly important when there is a need for people with diverse background and subtly different cultural assumptions to be working on and evaluating a project.

Just as a software product must adapt to diverse cultural contexts, so too the software development process will need to encompass the range of backgrounds participating in the development. Quality control in particular might involve obtaining feedback from individuals with different cultural conventions, and this is likely to be reflected not only in different perceptions or reactions for an image or a page layout; it may also appear in the way those reactions are reported.

Such variations in perception and semiotic interpretation are considered further in the following section.

4. Cultural Adaptation

As noted earlier, the problem of adaptation of software according to culturally specific interpretations of aspects

such as colour and sign is inherently complex, and difficult to automate. Such progress as has been made is based largely around the use of international standards and careful involvement of people native to the target environment. Given our focus upon the current research agenda, our treatment of this area will be somewhat cursory. Nevertheless, some coverage of the topic is an essential prerequisite for discussion of curriculum development and we shall proceed as follows.

User requirements based on cultural background are sometimes distinguished as either *overt* or *covert* factors according to subtlety (Yeo 1996)[4]. In the present work, overt cultural factors such as locale specific date formats have been considered as part of the internationalisation mainstream, and we are here concerned with the more difficult covert factors.

Covert factors normally fall into four subcategories:

- Mental Disposition: Differences in the mental disposition of people from different cultures induce consequent differences in their user interface design preferences, concepts of usability and priorities for functionality of the software;
- Perception: Perception is here used to encompass the whole psycholinguistic apparatus of interpretation of metaphors and symbolic representations (including colour, choice of icons, graphical art work, and audio signals);
- Social Interaction Rules: Social interaction rules are related to perception, but refer more specifically to the conventions governing interpersonal communication, through verbal dialogues, hand gestures, body language and facial expressions; and
- Context of Use: This refers to the environment within which the software is used – both the physical work space and the organisation in which it is deployed[5].

These factors have a enormous impact upon the design of the user interface for software products, with some common colour schemes and iconic representations proving intuitive within one culture but profoundly unsuitable within another. There has been substantial research on the impact of culture on user interface design (for example, Fernandes (Fernandes 1995), Ito and Nakakoji (Ito & Nakakoji 1996), Herman (Herman 1996), Yeo (Yeo 1996)). However, in practice, the attention paid to cultural variation in software design remains limited. The lack of awareness of cultural factors by software designers and the high set up and training cost have been perceived as two main reasons for this neglect.

While we would argue that interdisciplinary research in this area involving computer scientists, perceptual psychologists and cultural anthropologists is highly desirable, there is little doubt that such an enterprise will bear commercial fruit only in the longer term. From our perspective, adaptation to incorporate covert cultural factors must be viewed as a software engineering design and quality assurance problem – and in the absence of generic models, there can be no substitute for the technically literate native of the target environment.

However, these issues differ from string translation in that cultural expertise is required right from the outset of the requirements phase, in order to shape the overall architecture of the user interface, and not merely during adaptation and testing. The cost of adaptation may be markedly reduced if the software architecture is constructed with the specific aim of cultural adaptation in mind, and if pertinent information is archived and retained for future development. Common repositories and standardised international conventions would also assist toward this end, and this leads naturally into discussion of the standards movement, the topic of the next section.

5. Internationalisation Standards

As in the previous section, we begin with the caveat that our treatment of standards for software internationalisation is somewhat limited, and devoted more to a survey of those aspects of the problem in which standards have already appeared or are well-advanced – thus identifying constraints on possible research projects and source material for curriculum development. We are particularly concerned with the effectiveness of the standards movement in providing a framework for uptake of internationalisation by SMEs and in that respect the picture is somewhat disappointing – although there are some useful exceptions such as the AFSIT *Data Book* discussed below.

International standards pertinent to software internationalisation have emerged primarily through the explosion of interest in email and Internet services, with developments by numerous Working Groups and Technical Committees over the past decade. The International Organization for Standardization (ISO) is involved in a range of aspects, particularly at the lower levels of defining character sets and APIs. In Europe, the Trans-European Research and Education Networking Association (TERENA) has a working group WG-i18n considering the matter of internationalisation and the development of standards for inter-operation of services using multi-lingual documents, and the European Committee for Standardization (CEN) has a technical committee TC 304 on European Localisation Requirements, which deals with matters relating to character processing, such as keyboards and data entry, character coding and ordering, and various formatting conventions.

Internationalisation is a matter of special importance for interactions between Asia and "Western" nations; there is a greater gap in language and culture than occurs, for example, within Europe. The Asian Forum of Standardization for Information Technology (AFSIT) has set up a special interest group on Internationalization that takes a broader view of the issues. They have produced a *Data Book of Cultural Convention in Asian Countries* (CIC 1997) as a record of their activities. This resource addresses a range of matters including local conventions for numbers, dates, times, fonts, paper sizes, forms of address, measurement systems, colour significance, and taboo items. It is intended as a source of information, rather than a standard and forms a useful basis for curriculum and practice alike.

[4] Other, finer, sub-categories based upon functionality have also been proposed (Mahemoff & Johnston 1998).

[5] Arguably, such issues might include regional variations in computer hardware and telecommunications infrastructure – a problem of particular importance to the web design community due to bandwidth limitations. However, localisation of the minimal system specification is unlikely to be acceptable to software vendors.

The most fundamental problem addressed by standards is simply that of encoding different languages. Since the mid-eighties, ISO has been involved in developing character sets for different languages, beginning with the sixteen encodings of ISO 8859 series of 8-bit character sets. While these are mostly variations on the Latin alphabet, they also include Cyrillic, Arabic, Greek, Hebrew, Celtic and Thai.

This approach is inadequate for the pictographic languages, most notably Chinese, where thousands of characters are required, or the Korean Hangul script, which uses a phonetic alphabet, but arranged in two dimensional form that makes it more convenient to represent each complete syllable as one "character". The solution is Unicode; a universal character set that is able to represent every character in every language in common use today. The Unicode Consortium maintains the standard (The Unicode Consortium 2000), in close co-operation with ISO[6].

These various standards have now been integrated to the extent that all of the ISO 8859 series now map to Unicode – a transition readily exploited by Java's internationalisation support, through locale-specific character selections based around its native Unicode encoding. Adoption of the Unicode standard by SUN has markedly increased the visibility of internationalisation among information technology professionals and students, although many systems developed in Java run on platforms which do not use Unicode and so access to fonts is limited. Useful guides for the application of Unicode are provided by The Internet Mail Consortium (Internet Mail Consortium 1998), as part of their move to support its adoption.

Similar leadership has been provided in the web domain by the W3C through the development of version 4.0 of the HyperText Markup Language, HTML 4.0 (W3C 1999). This standard has been designed to support internationalisation of the World Wide Web, and while it mandates Unicode, it is principally of interest to us through its iniatives in support of cultural conventions, such as the direction of flow of a text. Other proposals for the internationalisation of URLs have been around for some time, but remain contentious (Yergeau 1996), and at this point, native users of the Latin character sets enjoy a distinct advantage.

While we have earlier discussed internationalisation support provided by modern programming languages and operating systems, these developments are indicative of the success of the standards organisations, from the introduction of locale by the POSIX (Portable Operating System Interface) standard (The Open Group n.d.), to its successors in the ISO umbrella project *Functionality of the internationalization of applications* (working group JTC1/SC22/WG20; the overall framework is described in (International Standards Organisation 1998)):

- ISO/IEC 14651. *International String Ordering.* This was published as a standard in 2001. It defines ordering on strings, and means to specify local orderings (International Standards Organisation 2001*b*).
- ISO/IEC DTR 14652. *Specification Methods for Cultural Conventions.* (Approval gained in June 2002 for publication as a technical report.) This report will not be a standard, and it is controversial in some aspects; but it will give useful guidance for specification of locale specific conventions for dates, times, money, address formatting, character ordering, and so on (International Standards Organisation 2001*a*).
- ISO/IEC WD 15435. *Internationalisation APIs.* (Working draft, but to be withdrawn.) This was considered as a possible standard for APIs relevant to conventions covered in the 14561 and 14562 documents. It is likely that a new project will be considered as a replacement.

Such work on standards is at once significant *and* elementary. The role of standards is to establish consistency of approach; and they may establish a preferred way of capturing and storing local conventions, which may be accessed by globally aware applications from different vendors.

This can have several benefits. It may limit the need for individual software houses to re-invent the wheel; it may permit applications to be tailored to a location without the need for recompilation; it may ensure that different applications are consistent in their behaviour; and it provides the possibility for refinment of local descriptions even after deployment of the application.

However, at present, standards do not adequately address the subtler aspects of cultural convention, and despite some honourable exceptions, the documents fall well short of comprehensibility necessary for immediate use as part of the developer's professional library. To the extent that such guidance is not provided by the standards organisations, the role of educating the profession and its apprentices must be taken up by the academic community, and the consequences of internationalisation for the curriculum are considered in the next section.

6. Curriculum

It is now apparent that academic interest in internationalisation is growing, with a number of universities across the world providing individual subjects and courses in the area. What is less clear is the extent to which internationalisation has transcended the boutique offerring to become an integral part of the curriculum, part of the knowledge base and associated skill set without which an information technology graduate would not be complete. For example, the joint IEEE-CS and ACM curriculum task force final draft of the CC2001 Computer Science Curriculum (The IEEE-CS and ACM Joint Task Force on Computing Curricula 2001) limits internationalisation to the core areas of social context (p.143), intellectual property (p.144) and privacy and civil liberties issues (p.145), with internationalisation in the present sense not even listed as an elective. Patently, some evangelism is required – although one may argue that internationalisation is less of an imperative for the American industry than for Australia and New Zealand.

Nevertheless, it is likely that those academic departments wishing to incorporate internationalisation issues will have to introduce the material gradually, and some possibilities are as follows:

Advanced Elective Units: A degree course might include elective units that focus on specific areas relating to internationalisation and localisation. The primary danger

[6] The character set of the Unicode standard 3.0 is identical to the standard ISO/IEC 10646-1:2000, and the two organisations maintain close links to ensure ongoing compatibility. The Unicode standard is the more comprehensive reference, as it addresses also a number of semantic properties for characters and character sets, like the direction of a script, orderings, and mappings between related characters.

in this approach lies in the potential for conflict between student expectations and the available time. In particular, students expecting a toolkit of techniques may be deeply disappointed by a wide-ranging, but relatively superficial overview. The apparently fatal pre-requisite of foreign language experience may be overcome through the use of a highly artificial pseudo-language, in which the issue is the demonstration of the technical problems of expansion and font selection rather than those of semantic equivalence.

Elective units of this nature may be hosted in a variety of subject areas:

- Software engineering for internationalisation could cover matters such as appropriate architectures, tool support, testing, support for diverse character sets, and resource files;
- Localisation of software and documentation could cover third party translation, planning for ease of translation, handling of locale-specific regulations or other semantic processing, and cultural aspects to localisation;
- While not *software* internationalisation in the strict sense considered here, a unit based around web site development could cover multiple language support, internationalisation of URLs and HTML, programming of locale-aware applets or other web based software, and browser-specific considerations.

Introductory Concepts in a Foundation Unit: While a basic introduction of this kind is helpful, and certainly better than no exposure at all, it is clear that its usefulness will be limited unless the concepts are reinforced as part of more advanced material.

As an Example in More Advanced Units: Internationalisation is ideal as a motivational example in general software engineering units for such basic concepts as separation of concerns, and maintainability and flexibility of software. Students can see immediately and in very concrete terms that these important engineering considerations do not merely give a benefit with respect to nebulous measures such as "quality"; it is simple economics in a global marketplace that such factors will determine whether or not a developer is able to compete and expand.

Threaded through the Curriculum: A general curriculum that is internationalisation-friendly might also include the use by default in all units of wide character sets, and the use of resources in larger software projects. Naturally, such an approach will limit the choice of programming language and environment. Matters relating to internationalisation or localisation should also be explicitly addressed in units that focus on the related topics of quality assurance, systems analysis, and World Wide Web development.

Discussion: In summary, the simple fact that an educational institution sees a role for such units in its offerings will have a beneficial effect on all students, even those choosing not to take such units, by raising awareness of the area and ensuring that graduates are aware that this is a matter of importance in information technology.

While the academic computer scientist and software engineer is properly focused toward the education of the next generation, university staff have a wider responsibility to engage in debate across the wider profession. Software internationalisation is an area of profound importance for the Australasian software industry, and one in which our activities may lead to wider acceptance of this imperative. Our own experience suggests that industry awareness of the importance of these issues is growing, and that an important aspect of the development of the curriculum is a healthy, bidirectional relationship with industry in which both parties benefit from new ideas they would not otherwise have encountered. A number of major vendors and institutions are already performing high quality work of this nature within the region, and we conclude in the following section with some disciplined speculation about the future of this cooperative effort.

7. Conclusions

In this paper we have examined a number of the core problems of software internationalisation at the present time, with a particular focus upon the opportunities for research and curriculum development. Given the commercial importance of the issues, however, it is critical that these opportunities be pursued in co-operation with the industry, cognizant always of the objective of economically sustainable adaptation of successful software to multiple environments.

The Australasian region is remarkably well-placed to host ongoing advances in this area and to foster a wider awareness of linguistic and cultural variation. Indeed, the presence in Brisbane of the Asia Pacific engineering centre for Red Hat, and a core development centre for Oracle – each with a strong internationalisation focus – is clear evidence for this proposition. We have the benefit of a common tradition of language and culture with the dominant English and American culture of the modern world, and we have the benefit of being a melting pot of many diverse and vibrant cultures and language traditions. We have a generally high standard of living and good access to education and to modern technology. We have demonstrated ability in innovation. For the most part, we have a good reputation overseas (though it would be unwise to presume on that too easily). We are a part of the burgeoning region of the Asia-Pacific, filled with expertise and energy and potential.

The global software market is an area in which Australia and the Asia Pacific can hope to make a significant and innovative contribution. There are many facets to the problem which are worthy of investigation, and there are opportunities and markets where internationalised software should do well.

In the various aspects of internationalisation reviewed in this paper, we can see ample scope for the researcher to pursue both intellectually interesting and commercially significant problems. While there is a solid foundation of work in translation and cultural adaptation, both fields are far from exhausted and we have identified in this paper the issues of string management and text generation from simplified sources as the most pressing research problems in the translation domain. With respect to languages and architectures, the core problems lie in the documentation of best architectural practice, and the integration of internationalisation workflows within software engineering processes – with the latter to incorporate some solution to the ongoing complaints of translation staff that they are given too little information and too little time to perform their task adequately. Such issues lie at the heart of good quality assur-

ance, and formalisation of these responsibilities is an important step in reducing internationalisation related defects.

In longer term, the international standards movement offers some hope for a solution to the most difficult of internationalisation issues, those of a general purpose internationalisation architecture – possibly through standardisation of the interface for internationalisation specific software components – and the specification of a framework for handling variations in covert cultural factors.

But it must be remembered that standards are not a panacea, and that such relief as they provide may often be a long time coming. Developers should take advantage of what standardisation exists, and then focus on making a success with the rest; standardisation has a history of following success, rather than the converse.

The most important focus, however, is for developers to put what is known into practice, and for students to hear about the issues in their University courses. We already have an excellent international mix in the Universities; and all the students, both visiting and local, can't help but be aware that users of information technology are from many diverse backgrounds. Enterprising students may even recognise the opportunities to establish links with one another that can pay off in the future.

What we need now is to place the problems and benefits and skills associated with producing internationalised software before those students by giving it serious examination as a demanding and profitable technical subject. They will find better solutions.

References

CIC (1997), 'Data book of cultural convention in asian countries'.

Dohler, P. (1997), 'Facets of software localization', *Translation Journal* **1**(1). http://accurapid.com/journal/01index.html.

Fernandes, T. (1995), *Global Interface Design*, Morgan Kaufmann.

Goo (n.d.), 'The Google language tools page', http://www.google.com/language_tools?hl=en. Accessed: June 30 2002.

Herman, L. (1996), Towards effective usability evaluation in Asia., *in* J. Grundy & M. Apperly, eds, 'Proceedings of OZCHI 96', IEEE Computer Society, Los Alamitos, CA, pp. 135–136.

Horstmann, C., & Cornell, G. (1999), *Core Java 2, Vol II, Advanced Features*, Prentice Hall, Upper Saddle River NJ.

International Standards Organisation (1997), 'Documentation – Presentation of translations'. ISO/IEC 2384.

International Standards Organisation (1998), Information technology – Framework for internationalisation, Technical Report ISO/IEC TR 11017:1998, ISO.

International Standards Organisation (2001*a*), 'Functionality for internationalisation – Specification of cultural conventions'. ISO/IEC DTR 14652.

International Standards Organisation (2001*b*), 'Information technology – International string ordering'. ISO/IEC 14651:2001.

Internet Mail Consortium (1998), 'Using international characters in internet mail', IMCR-010. http://www.imc.org/mail-i18n.html.

Ito, M. & Nakakoji, K. (1996), Impact of culture on user interface design., *in* E. M. del Galdo & J. Nielson, eds, 'International User Interfaces', John Wiley & Sons, New York, pp. 105–126.

Kokkotos, S. & Spyropoulos, C. D. (1997), An architecture for designing internationalized software, *in* 'Proceedings of the 8th International Workshop on Software Technology and Engineering Practice (STEP)'.

Koomen, T., Pol, M., Broeders, H. W. & Voorthuyzen, H. (n.d.), *Test Process Improvement: A Practical Step-by-Step Guide to Structured Testing*.

Lan (n.d.), 'Language partners international, improving results with machine translation software', http://www.languagepartners.com/reference-center/whitepapers/mtwp/mtartic.htm. Accessed: June 30 2002.

Lerner, M. (1999), 'Building worldwide web sites', IBM DeveloperWorks. http://www-106.ibm.com/developerworks/library/web-localization.html.

Mahemoff, M. J. & Johnston, L. J. (1998), Software internationalisation: Implications for requirements engineering, *in* D. Fowler & L. Darson, eds, 'Proceedings of the Third Australian Workshop on Requirements Engineering', Geelong, Australia, pp. 83–90.

Mollá, D. & Schwitter, R. (2001), From plain English to controlled English, *in* 'Processing of the 2001 Australasian Natural Language Processing Workshop.', Sydney.

Paris, C. e. (2002), 'The Isolde project'. http://www.cmis.csiro.au/iit/Projects/Isolde/index.htm.

Paulk, M. C., Curtis, B., Chrissis, M. B. & Weber, C. V. (1993), 'Capability maturity model, version 1.1', *IEEE Software* **10**(4), 18–27.

The CMMI Product Team (2002), CMMISM for systems engineering/software engineering/integrated product and process development/supplier sourcing, version 1.1, continuous representation, Technical Report CMU/SEI-2002-TR-011, SEI. (CMMI-SE/SW/IPPD/SS, V1.1, Continuous).

The IEEE-CS and ACM Joint Task Force on Computing Curricula (2001), 'Computer science volume, final report', http://www.computer.org/education/cc2001/final/index.htm. (December 15 2001).

The Open Group (n.d.), 'Locale'. http://www.opengroup.org/onlinepubs/007908799/xbd/locale.html.

The Unicode Consortium (2000), *The Unicode Standard, Version 3.0*, Addison-Wesley, Reading, MA, USA. **URL:** *http://www.unicode.org/unicode/uni2book/u2.html*

W3C (1999), 'HTML 4.01 specification'. http://www.w3.org/TR/1999/REC-html401-19991224.

Yeo, A. (1996), 'World-wide CHI: Cultural user interfaces, a silver lining in cultural diversity.', *SIGCHI* **28**(3), 4–7.

Yergeau, F. (1996), Internationalization of URLs, Technical report, Alis Technologies. http://www.alis.com:8085/~yergeau/url-00.html.