An essential first step in the localisation of software is the proper encoding of the scripts of the new language. Drawing upon experience in Saudi Arabia in the 1970s and now in Nepal, I set out a number of principles to guide the production of code tables, and then analyse alternative proposals for Nepali. From this it is shown that existing code tables in Unicode have shortcomings and should be reviewed, while the proposals under discussion for Nepali should be given time to be worked through.

It is desirable for computers to work in the scripts and languages of the users of the computers. A large proportion of the world’s population does not speak English. Even in India where English is a common lingua franca, only 5% of the population speaks English sufficiently well to use computers effectively. For these communities to benefit from computers to the extent that western communities have, their computers must work in their own local languages.

Computers were developed in the west, mostly in the U.S., and thus were developed to interact with their users in American English. But as costs have plummeted and capabilities risen, computers have become widespread commodities so that more than half the sales of computers today are outside the U.S. (eg. Taylor, 1992; Hall and Hudson, 1997). This in turn has led to pressure for localised versions of software. Suppliers of hardware and software have responded to this demand, though not always with the willingness that these markets would deserve.

During the late 1970s computers began to be introduced into the Arab world in large numbers, and slowly the Arab users began to ask that they should work in Arabic. The systems were supplied almost entirely by companies from Europe and the United States, who sometimes responded that it was not possible, and sometimes rose to the challenge. Their first challenge was to make devices that had assumed writing was from left to right to operate right to left, while the second challenge was imaging a set of characters that required detail of a much higher resolution, like dots above or below the characters, not required for the Roman Alphabet except for a few diacritics. The alphabet to be input and output could be found in any text book on Arabic, and it was assumed that providing that those shapes could be typed in and printed, all would be well.

But it was not, and naive computer professionals made serious blunders—the most singular of these was done by a well known UK company operating in Kuwait who noted that there was a circular shape like an “O”, but a little more egg shaped, which they then gave a single code and a single place on the keyboard - in fact there were two different characters of Arabic involved, one being the number five (Khamza) and the other a special letter Teh Marbuta used at the end of words (strictly written with two dots above it which are often omitted).

This situation in Arabic computing led me and several colleagues to investigate the whole issue of how Arabic should be handled in the computer (Hall and Hussein 1978, Hall 1978, Hall and Waghorn 1979, Hall 1981) leading a number of us active in the informal Riyadh Computer Users Group to make a proposal for a set of Arabic codes which we offered to...
the Saudi Arabian Standards Association. In drawing up this proposal, we were influenced by an earlier proposal from Iraq which had decided that it was necessary only to code the letters and not the forms of the letters, an idea which had also been investigated in Lebanon. This led to a patent for a simpler form of typewriter in which the typist only selected the letter needed and not the form of letter, leading to significantly higher typing speeds. Both these proposals assumed an intelligence in the computer which could work out which form of letter to use following some simple rule. However, there were some problems, and the rules were not that simple for cases like Hamza (a glottal stop always written as a small diacritic but on a “carrier” which varied following rules determined grammatically).

Our ad hoc standards drafting committee built on this earlier work and implicitly used a set of design rules to determine the code tables of our proposal, but we did not feel it appropriate to publish those rules. Now, twenty years later, I am working in Nepal and find a situation not unlike that I found in Saudi Arabia in 1978. There is a plethora of code tables, and an urgent need to bring coherence to these, before too much data gets stored and then needs conversion before it can be shared. In the process, we will have to undertake a systematic design of the code tables for Nepali, and those rules used implicitly for Arabic are going to be important.

The objective in this paper is to set out those rules, suitably updated and improved by knowledge acquired since. Then those rules will be applied to codes for Devanagari script, and in particular the use of the script for the Nepali language. Then, when at some later time, some new writing system must be supported by computer, these rules can be applied to determine what should be coded, and how it should be coded. It is also suggested that existing codes should be fully reviewed.

Basic Concepts and Terms

When we look at computers and their use in a particular language, what we see are the input devices, typically keyboards, and the output in the writing system of the language on the screen and on paper. It is very tempting to focus on these and ask what characters should be printed, and where should these appear on the keyboard for input. But the situation is much more subtle than this, and we will need to distinguish a number of concepts important in the area, but which may be confused in everyday discourse. We will go through these terms, turning both to the Concise Oxford Dictionary for general use and to the Penguin Dictionary of Language and Languages (Crystal, 1992) for linguistic usage.

First, we need to distinguish between a language and the way that it is written, using a script (the collection of graphic symbols known as graphemes or glyphs) and an orthography (which includes the script and the prescribed spelling and punctuation rules). The same script, or essentially the same script, may be used for many languages, as in the European languages most of which use the Roman script. These languages do not even need to be of the same general structure (linguistic group) as in the use of Devanagari (Brahmi) derived scripts used for languages of the Indo-Aryan group (where these scripts originated) but also for Tibeto-Burmese languages (e.g., Tibetan and Thai) and Dravidian language (e.g., Tamil). Equally well, the same language may be written in many scripts, as has happened historically with Turkish which switched from the Arabic script to the Roman script in 1928 as part of the “modernisation” programme of Kemal Ataturk. It does happen in Japanese with the ideographic Kanji writing system and the Kana syllabaries Katakana and Hiragana and the proposed (Matuura, 1997) Romanisation of Japanese.

Scripts can be broadly of two types—concept and phonemic. Concept systems typically use ideographs, each ideograph representing some concept in the world of the users - typically words in other systems: the archetypal concept system is Chinese, where the written form is intelligible across most of China whereas the spoken form is not; pictogram systems are also concept systems, the most well known being the ancient Egyptian hieroglyphs from 3000B.C. “Concept system” is not a term used by linguists, though Crystal has the entry for ideogram as “A symbol used in a writing system to represent a whole word or concept,” but it does seem to be a useful term.

Phonemic systems concentrate on the spoken form of the language and aim to encode that form—or at least they originally had that intention. Concept systems lead to a very large set of ideographs, while phonemic systems are very much smaller.

Phonemic systems are to two types: alphabets in which letters represent single phonemes, or syllabaries in which the characters represent syllables, typically consonant vowel pairs. Alphabets are by far the most common, typified by the Roman and Arabic alphabets: the syllabary that is viewed as typical is the Kana system of Japan. Not all experts agree on where to place a script—so for example, the Devanagari system is claimed by some to be an alphabet in which the letters can take different forms depending upon their position in the word, so that vowels may be recorded as signs in the middle of words, and consonants may be represented as partial characters when used medially in conjuncts; while others view Devanagari as a syllabary in which the base characters are the composites of consonant plus vowel sign and the conjuncts. Typically syllabaries are significantly larger than alphabets, as will be appreciated from the combinatorics of the situation (if you have 20 consonants and 5 vowels, you have an alphabet of 25 letters, but a consonant-vowel syllabary of potentially 100).

Writing Systems

Sometimes the term “writing system” is used as a synonym for orthography (e.g., Crystal, 1992) but we will distinguish it and use writing system in a more comprehensive technical manner to encompass the processes of reading and writing as well as the script with its glyphs and graphemes, and the orthography prescribing rules of combination in sequence...
within the language. We must recognise that the human users of these must write the written form in some manner to give a permanent or semi-permanent record, which can then be read by some other human, or indeed that same person later. With computers, this process becomes very visible with keyboards and other means for text input from the human (= writing) and display screens and printers for output to the human (= reading).

We must also distinguish the styles and aesthetics of writing from the scripts. Any script can be written in a variety of styles - the precise style is determined by cultural and aesthetic factors: in print, the style is determined by the selection of a particular font, in writing it is determined by calligraphy. In many cases, the intention is to communicate the text and the selection of style is of second order significance, though often changes of style within a passage of text are intended to signal meaning - as in the marking of headings and subheadings or emphasis in documents. We will take the view that all these matters of style and formatting lie outside the script and are part of the particular computer application.

In some cases, the characters of a language are used as elements in some design and in these cases we will not consider that the use of the characters has any technically meaningful relationship to the script or language. Examples of this are seen in all languages and scripts, but reaches a particularly high art in Islamic calligraphy. Of course these are meaningful, and articulate parts of sacred texts like the Koran. This meaning will not be used within the computer, so the calligraphic design can be viewed as a picture or graphic rather than as text, and the internal characters within the design will not need to be separately manipulated.

Now in the very simplest form of writing system the human would select a character (glyph or grapheme) from a fixed repertoire, say by selecting a key to press on the keyboard, and this would be presented back to the human on the display screen or printer. A single action inputs the character creating an electronic representation of it which is then stored internally using some code. There is a one-to-one correspondence between input actions (key choices), internal codes and glyph displayed on the screen. This was the case in the earliest computers, but this soon became complicated by shift keys to extend the range of the keyboard (a natural thing to do with the precedent of the typewriter). But as the repertoire for input increases further, a single shift will not do — this is markedly the case for concept scripts with their very large repertoire of characters. On the input side not just a single key or combination of two keys was sufficient, but multiple key combinations or sequences of keys are also needed.

By symmetry on the output side, it could also be possible to choose the glyph to be displayed or printed as a function of several stored characters. This was found to be necessary for one representation of the Arabic script which needed the characters on either side of the current character to be able to determine the actual glyph to be used.

Further, with computers we need to recognise that the computer applications that act upon the text may have their own special requirements, as for sorting, and would produce their own representations of the text as required. This leads to the flow diagram of Figure 1 for the writing system of a script and language using the computer.

Let us illustrate this with a few examples. One method of inputting an ideographic script is to use a stylus and tablet, and note the “brush” strokes of the human and from the sequence, direction and length of these determine the actual
character written within the input mapping routine - a form of character recognition - see Figure 2. Similarly for more conventional alphabets an alternative to pressing a combination of keys (like shift plus one other) together is to press them in sequence: examples of this appear in Microsoft’s Word program where the sequence “(, “c”, ”)” is interpreted as © which is then stored.

An example of an output mapping routine occurs in Arabic where the form of each letter may vary depending upon whether the letter appears at the beginning or end of a word or in the middle or in isolation. Only the letters and not their forms are stored, and on output the form to be displayed or printed is determined by a very simple algorithm from the letters on either side.

It may seem that putting in extra processing at output would cause special problems, since many applications want to know how much space on the screen or page the text will take. It could seem that having a one-to-one correspondence between internal codes and printable characters would be essential. However, even in this case, the application must ask the print or display subsystem for space guidance since characters will be proportionally spaced, and may vary in their kerning (the extent to which the characters overlap so, for example, in some fonts the bottom of a “t” might curl beneath the top of a following “w”). If the print/display subsystem also stacks characters vertically, this becomes more complex - but is not different in kind. Let us look at an example of stacked characters, this time taking the example from Devanagari, as in Figure 4.

The applications can be very variable, but what they actually do with the text is limited:

- matching substrings for identity or equivalence or possibly approximate matching (see Hall and Dowling 1980)
- sort the strings or substrings, using some principle to induce a total ordering upon them.

following which some rearrangement may take place. Often these operations mean working through the text character at a time, but this may not always be the case. The key concerns here are equivalence, and sorting.

It is possible that two items of text are not identical, but are equivalent. This can happen with alternative spellings of the same word (which often happens with proper names, and in German where an umlaut over a vowel is often replaced by an “e” after the vowel), but it can happen as an artifact of the encoding. For example, if in Arabic we coded the forms of letters, then should we accept as equivalent any choice of form? The response here is that we should. But we could change our encoding so that such artifactual equivalences do not arise.

For sorting we may need to consider not just the characters one at the time, but several characters at a time - so in sorting Spanish, “ll” is treated as a single letter to come between “l” and “m” (del Galdo, 1990). In the Roman script, capital letters and small letters are considered equivalent for sorting purposes. This means that we must either extract a sort key by an algorithm in which the rules of comparison are coded, and then to actually rearrange the text using the sort key.

Figure 2. Two Chinese characters, one simple “single structure” and one complex “multiple structured, showing a stroke analysis after the methods of Li and Li (1985), that might be used for input.

Figure 3. The Arabic alphabet has 28 letters, each of which can take up to four forms, depending where in the cursive flow of the writing it occurs - at the start, in the middle, at the end, or isolated and standing on its own.

Here are three of the letters that take all four forms, though many of the letters only take two forms.

Figure 4. A composite vertically stacked character in Devanagari.
and simple comparisons; or we must encode the rules directly in a string comparison routine. Certainly we cannot expect to be able to select byte encodings such that sorting can be done by simple byte comparisons.

**The Code Design Problem**

This now gives us a complex design problem with many degrees of freedom. For example, if we wish to type a character with a diacritic, we could type the character then the diacritic and let the input routine resolve this to encode the result as a single internal code, which prints directly. Or we could store the two characters and resolve them at print time to image as a single character accompanied by the diacritic. How do we decide?

The central representation of the text is that internal to the computer, and using this, we must be able to undertake a variety of applications and be able to display and print the material in a manner that human readers find comfortable. Furthermore, we must be able to input text into this internal representation and equally importantly be able to change this text mediated by its output representation. What we need is an internal representation that is more abstract and captures the essence of the text. This is a strategy that we frequently use in computing. In data modeling, we capture “conceptual schema” that can be independent of any physical implementation. In programming languages, we represent the essence of the language by an abstract syntax which shows the basic constructs and their sequence, but is devoid of any “syntactic sugar” to enable parsing or match the particular needs of particular uses.

The general principle is that we should encode what is needed and no more. Extra codes take up more space in the code table, though they may help compact the storage of any particular tract.

How do we determine what is needed? We shall use the method of contrastive units - when we make a change, does it make any difference?

Now let us work through a larger example from Arabic, looking at the Arabic word for Arabia, “Al Arabia.” This would normally be written as shown in Figure 5(a) from right to left. This word is made up of the sequence of letters shown in Figure 5(b). This time the sequence is shown in the sequence of English, from left to right with letters shown in their isolated form. Our problem is, do we encode and store the letters or the forms of letter: alif, lam, ain, reh, beh, yeh, teh. Now the form is determined entirely by simple rule, with the exception of the final teh marbuta where the distinction between a final teh and a teh marbuta is complex.

How do we decide in general terms on the essential abstract form? The contrastive unit approach suggests systematically substituting one element by another and determining whether it makes a difference. So in Arabic if we substitute one form of a letter by another form, does it make a difference? To a calligrapher it might - it could be a matter design to select the inappropriate form, and a calligrapher might do this for

---

**Figure 5.** The design choices for encoding Arabic (see text)

---

**Figure 6.** The Unicode table for Arabic, lower half only (upper half includes extensions to enable the writing of other languages): source - http://www.unicode.org/Unicode.charts/glyphless/U0600.html
effect - but to the majority it would just be wrong (a kind of misspelling) but understandable. In our earlier discussion I suggested that issues of artistic design should be set aside, and hence we should go with the majority - the essence of the script lies in the letters and not the forms of the letters. This would go not just for the majority of the letters, but also for the 'troublesome' ones of the teh marbuta, alif maksura, and especially the hamza, though the rules for these are more complex and need further investigation. If we now look at the code table for Arabic in Unicode, shown in Figure 6, we see that in most cases it is the letter that is coded, except for except for those troublesome characters (teh marbuta at 0629, alif maksura at 0649, and especially the hamza at 0623 to 0626), and possibly some of the diacritics. It is clear that the Arabic code table needs proper and informed review.

We see then a design process that consists of the following steps:

1. Determine the full set of characters that are written or printed.
2. Eliminate redundant ones to obtain a set of independent characters, using the principle of contrastive units.
3. Determine other languages that use the same script and look for common ground - letters and conventions used by large sets of languages that use the script.
4. Encode this character set taking into account other languages that use the script, adding also the standard control characters, numerals and punctuation and other symbols, choosing codes for special symbols (like numerals) that coincide with other code tables.
5. Determine an input mapping - determine places on the keyboard and combinations and sequences of key strokes to input the characters.
6. Determine an output mapping, to image the characters or combinations of characters in the manner expected by users.

### Application to Nepali

Nepali is a language of the Indo-Aryan family that is written using the Devanagari script. The use of computers in Nepal is relatively recent, and at present there is no agreed encoding of the scripts: indeed the need to do so is only now...
The first computer was imported into the Kingdom in 1974, an IBM 1401 computer to help with the analysis of the 1971 and 1975 censuses. This worked entirely in English. In 1981, a start-up company, Innovative Computers, imported the first Apple IIE and developed a font and encoding for this. They attempted to copyright these fonts, and were instead granted a Nepali patent. In 1983, an electronic engineer, Muni Sakya, developed special hardware for the Devanagari font. Subsequently Devanagari has gradually been accepted for applications like word processing, and each supplier of computer equipment now creates their own code table and font, feeling obliged to offer some support for Nepali, but differentiate their offering from that of their competitors. Up to the present there has been no demand for document or other exchange of files in Nepali, and hence the incompatibility between the different suppliers of computer systems has not yet created problems.

The encodings fall into broad groups, anecdotally having been derived from a common source where suppliers have made small changes to differentiate their product. Two typical representatives of commonly used groups are shown in Figure 7.

The main differences between these tables lie in the placement of the numerals in the tables - positions 33 to 41 on the left, and positions 48 to 57 on the right to gain compatibility with ASCII and help the use of spreadsheets and similar. Preeti on the left is an earlier table (1993) and has far fewer characters than Himali New (1995) on the right where the creators of table have added more composite characters and alternative forms, for example to allow the proper placement of diacritic marks. Currently Preeti is being used as the primary coding table and font set for the important Nepali Dictionary being prepared at the Royal Nepal Academy.

It is claimed that there are no standards in place, and hence the diverse practices. And yet Nepali is written in a derivative of the Devanagari script used for Hindi, and Devanagari is represented in Unicode, as shown in Figure 8.

The reasons for rejecting this table appear to be socio-political, to differentiate Nepal from its large and powerful neighbour, India. The evidence for this, and the consequences of this, are outside the scope of this paper.

Let’s now apply our rules for code tables to Devanagari and Nepali.

The Devanagari writing system, on which the Nepali writing system is based, is an alphabetic system (Crystal, 1992, 99); though some authorities claim that it is syllabic, such as Bandhu (1976). The descriptions I have seen have clearly made it alphabetic. That is, each phoneme of the language is represented by a distinct character or letter; though it is easy to see that the composite of consonant with following vowel shown as a diacritic could be viewed as a syllable.

As previously pointed out, most languages of South Asia use writing systems derived from Brahmi, from which the Devanagari system has been derived. Crystal (1992) gives a evolutionary tree for these scripts, shown in Figure 9. Nepali is not shown, but would fall under Devanagari in parallel with Gurmukhi.

The correct orthography of Nepali has been the subject of much debate as the body of material written in Nepali has risen sharply. Bandhu (1976) in his discussion of the correct spelling of Nepali has suggested returning to the origins of the writing system—the basic phonemes: There has been a proposal for a completely new writing system called Navanagari (Shrestha 19xx). But this would involve a radical change, and could only work in situations where the writing system was relatively young, or did not yet exist. For Nepali I will focus on the current writing system as is used in practice.

Nevertheless, the return to phonemes does suggest an important principle—the pursuit of contrastive units at the level of phonemes, of the spoken language. Which of the characters, forms of character, combinations of character, make a difference?

Matthews (1992) gives an alphabet of 11 vowels which
occur in two forms, as a full character and as a diacritical sign, and 35 consonants. Opinion seems to vary about the exact number of consonants that truly belong to Nepali, since some may be present to represent loan words brought in from Sanskrit (and Hindi or Urdu and from elsewhere).

Consonants that appear together can be written as “conjuncts,” as a composite character — Matthews (1992) lists 58 of these. Many of these are formed from medial forms of the consonants which are used in combination with other medial forms written in sequence, and this is how most conjuncts are handled on typewriters and in the code tables cited above. For example, you will see in the Himali New table pairs of characters first the initial and isolated form, and then the medial form for conjuncts - as in positions 81 and 82, 83 and 84, and so on. Conjuncts may also be stacked vertically, as previously discussed and shown in Figure 4.

We have thus far identified the repertoire of characters used in writing and print, and completed the first step of our design process. We now embark upon the second step - what is the essence of the Devanagari alphabet as used in Nepal, that we should encode and store? It seems that we have a choice, to encode the full alphabet and all conjuncts - or to follow the lead of Arabic and simple go for the letters of the alphabet and not their forms.

The current trend in discussion within Nepal aimed at establishing a standard, and aimed at supporting projects like the Nepali Dictionary, is to go for the full set of glyphs. Current discussions suggest a full set of 11 vowels in isolated, medial and sign forms, 36 consonants, 29 half consonants for use in conjuncts and 33 vertical conjuncts - a total of 127 characters. With another 39 numerals and symbols of various kinds for use in formulae, etc. this makes a total of 176 characters, well within the space available in an eight-bit encoding. Each character as represented is printed one-for-one, making a simple output system.

However, the input side has some difficulties. There are between 45 and 50 keys on a keyboard, and thus we would need two shift keys - usually these would be the Alt and the Shift keys pressed simultaneously with the character key, and in some cases, means pressing two keys with one hand. Some of the letters would need to be placed off the normal home position of 30 keys in the central three rows. This in turn leads to concern with typing speed, though the response to this concern at the moment is that people are pleased to be able to do it at all. A number of technical specialists in Nepal have been experimenting with sequential keying, using Word Macros like those used to turn “(“ “c” “)”) into “©” and the Keyman package so that, for example, a consonant followed by a “:” and then another consonant might mean join them horizontally, while a “/” between them would mean join them vertically. The claim is that this would increase typing speed, but at present this is an argument that is not found to be persuasive by the intended local beneficiaries. This approach seems to me rather like the stroke recognition approach to Chinese input.
with a typist building up the intended glyph in a painterly manner. But it could work, providing all the glyphs for output have been accounted for.

The alternative approach is to encode just the consonants and the long vowels. This gives a repertoire of 47 characters, which nicely fit into the home position of the keyboard requiring just a single shift. This clearly simplifies the input side, and with fewer choices for the typist should increase typing speed and accuracy. Some of the application processing, like sorting, will be simplified, but it complicates the output side. The code table could be that subset of the Unicode table with this minimal set of letters.

Of course we could develop a system with the simple direct keyboard of the second approach backed up by lot of intelligence at input to create the correct internal code from the first approach with its complete glyph set. But this in turn would lead to problems with editing - what you input and what you see is not what you get.

So which do we choose? The complexity of the third approach seems to make it unsustainable, while the first is “safe” in the sense it offers everything that current print systems offer by focusing on output. The second approach needs some further investigation. It is possible that the act of combining characters to form a conjunct actually declares a syllable and partially determines the meaning. I have been assured that this is not the case, but need to consult further experts.

If both first and second are sustainable, the way forward would seem to be to do some market testing. There has been talk of market testing keyboard layouts and the sequential versus simultaneous protocols, so this extra level of market testing could be feasible. That is the work of the next few months.

The third step of process has already been discussed above and shown in Figure 9, and the remaining steps of assigning codes and determining the input and output mappings are premature.

There is one difficulty in all this — Microsoft is rumoured to be preparing a Nepali version of Windows 98, and it could be that an industrial standard will preempt any more considered design of character set, code table and keyboard mapping for Nepali.

Conclusions

As the use of computers spreads, there will be an ever-increasing demand for the ability to represent information in local languages. This is being further fueled by the Internet, where usage in languages other than English is the fastest growing sector. In order to be able to represent information in these local languages, the writing systems for the languages must be encoded in the computer in some standard manner. It must be a standard to ensure that the exchange of information can be undertaken, and this standard must be informed by the best linguistic and technical knowledge available — the encoding will be difficult to change later.

In this paper we have seen a systematic procedure for doing this, for determining the internal coding of the new writing system, or for reviewing the existing codes of writing systems that have already been encoded.

We then illustrated this process for Nepali. In doing this we found that current approaches embodied in the code tables of Unicode standard (and indeed other standards) may not be optimal. We saw this for Arabic and for Devanagari, the writing system of Hindi, and this is a sufficient sample for me to feel unquiet about the encodings of the many other languages already covered by Unicode. Just what were the rationales for them?

In looking at Nepali, we saw a situation in a state of flux that might be about to be overtaken by commercial interests. I hope that the deliberations currently taking place in Nepal can be given time to complete, so that the interests of the people of Nepal can be properly served.

This hope expressed for Nepali is equally true of all languages - they need to be encoded in the computer taking into account the best interests of the community of users of the language. If the user community does not take control of the standardisation of the character codes for their own language, then commercial interests may do it for them. The result may well then serve commercial expediency rather than the user community.

Acknowledgments

I am grateful to a large number of people with whom I have talked while in Nepal carrying out this research. Some have been acknowledged through the references, but others need explicit mentioning. I would like to thank Allen Tuladhar, Bhanu Pathak, Churamani Bandhu, Dan McCloy, Jeff Rollins, Jyoti Tandukar, Nirml Man Tuladhar, Peter Malling, Postun Pradhan, Rajesh Shakya, Rajib Subba, Ken Keniston, Sunil Shrestha, Yogendra Yadava, and others whose names I failed to record. I apologise for any omissions.

References

Bandhu, Churamani (1976). Transcription and Orthography, Seminar papers in Linguistics, Institute of Nepal and Asian Studies (INAS), Tribhuvan University, Kirtipur, Nepal, 103-116


Hall PAV (1978). Man-Computer Dialogues for many levels of Competence, ECI78 conference, Venice, Oct 78, published by
Patrick Hall has been Professor of Computer Science at the Open University in the UK since May 1991, and previously at Brunel University from 1987. Prior to this, he held various jobs in industry concerned with the development of large systems, from database management systems to command and control systems, in companies from research organisations to product builders to software houses. His general research interests are in software technology, with major funded projects on testing and software reuse. A strong thread of research has been in software globalisation arising from a commercial assignment in Saudi Arabia in the late 1970s, and pursued since then in a number of projects across Europe and now in South Asia.
Related Content

Beyond Localization: Effective Learning Strategies for Cross-Cultural E-Learning
[www.irma-international.org/chapter/beyond-localization-effective-learning-strategies/8292/](www.irma-international.org/chapter/beyond-localization-effective-learning-strategies/8292/)

[www.irma-international.org/chapter/exploring-trust-building-mechanisms-global/19062/](www.irma-international.org/chapter/exploring-trust-building-mechanisms-global/19062/)

A Comparative Study of the Effects of Low and High Uncertainty Avoidance on Continuance Behavior
[www.irma-international.org/article/comparative-study-effects-low-high/42087/](www.irma-international.org/article/comparative-study-effects-low-high/42087/)

The Impact of Information Sharing on Order Fulfillment in Divergent Differentiation Supply Chains
[www.irma-international.org/article/impact-information-sharing-order-fulfillment/51323/](www.irma-international.org/article/impact-information-sharing-order-fulfillment/51323/)

Doing Business in Paradise: How Small, Information Intensive Firms Cope with Uncertain Infrastructure in a Developing Island Nation (TCI)
[www.irma-international.org/article/doing-business-paradise/3597/](www.irma-international.org/article/doing-business-paradise/3597/)